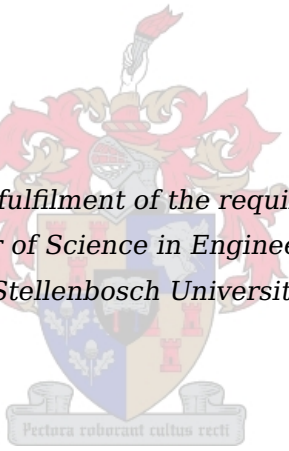
# Stereo vision for simultaneous localization and mapping

by

Wikus Brink




*Thesis presented in partial fulfilment of the requirements for the degree of*
*Master of Science in Engineering*
*at Stellenbosch University*

Supervisors:

Dr C.E. van Daalen
Electrical and Electronic Engineering

Dr W.H. Brink
Mathematical Sciences

December 2012

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted if for obtaining any qualification.

Date: December 2012

# Abstract

Simultaneous localization and mapping (SLAM) is vital for autonomous robot navigation. The robot must build a map of its environment while tracking its own motion through that map. Although many solutions to this intricate problem have been proposed, one of the most prominent issues that still needs to be resolved is to accurately measure and track landmarks over time. In this thesis we investigate the use of stereo vision for this purpose.

In order to find landmarks in images we explore the use of two feature detectors: the scale-invariant feature transform (SIFT) and speeded-up robust features (SURF). Both these algorithms find salient points in images and calculate a descriptor for each point that is invariant to scale, rotation and illumination. By using the descriptors we match these image features between stereo images and use the geometry of the system to calculate a set of 3D landmark measurements. A Taylor approximation of this transformation is used to derive a Gaussian noise model for the measurements.

The measured landmarks are matched to landmarks in a map to find correspondences. We find that this process often incorrectly matches ambiguous landmarks. To find these mismatches we develop a novel outlier detection scheme based on the random sample consensus (RANSAC) framework. We use a similarity transformation for the RANSAC model and derive a probabilistic consensus measure that takes the uncertainties of landmark locations into account. Through simulation and practical tests we find that this method is a significant improvement on the standard approach of using the fundamental matrix.

With accurately identified landmarks we are able to perform SLAM. We investigate the use of three popular SLAM algorithms: EKF SLAM, FastSLAM and FastSLAM 2. EKF SLAM uses a Gaussian distribution to describe the systems states and linearizes the motion and measurement equations with Taylor approximations. The two FastSLAM algorithms are based on the Rao-Blackwellized particle filter that uses particles to describe the robot states, and EKFs to estimate the landmark states. FastSLAM 2 uses a refinement process to decrease the size of the proposal distribution and in doing so decreases the number of particles needed for accurate SLAM.

We test the three SLAM algorithms extensively in a simulation environment and find that all three are capable of very accurate results under the right circumstances. EKF SLAM displays extreme sensitivity to landmark mismatches. FastSLAM, on the other hand, is considerably more robust against landmark mismatches but is unable to describe the six-dimensional state vector required for 3D SLAM. FastSLAM 2 offers a good compromise between efficiency and accuracy, and performs well overall.

In order to evaluate the complete system we test it with real world data. We find that our outlier detection algorithm is very effective and greatly increases the accuracy of the SLAM systems. We compare results obtained by all three SLAM systems, with both feature detection algorithms, against DGPS ground truth data and achieve accuracies comparable to other state-of-the-art systems.

From our results we conclude that stereo vision is viable as a sensor for SLAM.

# Opsomming

Gelyktydige lokalisering en kartering (*simultaneous localization and mapping*, SLAM) is ’n noodsaaklike proses in outomatiese robot-navigasie. Die robot moet ’n kaart bou van sy omgewing en tegelykertyd sy eie beweging deur die kaart bepaal. Alhoewel daar baie oplossings vir hierdie ingewikkelde probleem bestaan, moet een belangrike saak nog opgelos word, naamlik om landmerke met verloop van tyd akkuraat op te spoor en te meet. In hierdie tesis ondersoek ons die moontlikheid om stereo-visie vir hierdie doel te gebruik.

Ons ondersoek die gebruik van twee beeldkenmerk-onttrekkers: *scale-invariant feature transform* (SIFT) en *speeded-up robust features* (SURF). Altwee algoritmes vind toepaslike punte in beelde en bereken ’n beskrywer vir elke punt wat onveranderlik is ten opsigte van skaal, rotasie en beligting. Deur die beskrywer te gebruik, kan ons ooreenstemmende beeldkenmerke soek en die geometrie van die stelsel gebruik om ’n stel driedimensionele landmerkmetings te bereken. Ons gebruik ’n Taylor- benadering van hierdie transformasie om ’n Gaussiese ruis-model vir die metings te herlei.

Die gemete landmerke se beskrywers word dan vergelyk met dié van landmerke in ’n kaart om ooreenkomste te vind. Hierdie proses maak egter dikwels foute. Om die foutiewe ooreenkomste op te spoor het ons ’n nuwe uitskieterherkenningsalgoritme ontwikkel wat gebaseer is op die RANSAC-raamwerk. Ons gebruik ’n gelykvormigheidstransformasie vir die RANSAC-model en lei ’n konsensusmate af wat die onsekerhede van die ligging van landmerke in ag neem. Met simulasie en praktiese toetse stel ons vas dat die metode ’n beduidende verbetering op die standaardprosedure, waar die fundamentele matriks gebruik word, is.

Met ons akkuraat geïdentifiseerde landmerke kan ons dan SLAM uitvoer. Ons ondersoek die gebruik van drie SLAM-algoritmes: EKF SLAM, FastSLAM en FastSLAM 2. EKF SLAM gebruik ’n Gaussiese verspreiding om die stelseltoestande te beskryf en Taylor-benaderings om die bewegings- en meetvergelykings te lineariseer. Die twee FastSLAM-algoritmes is gebaseer op die Rao-Blackwell partikelfilter wat partikels gebruik om robottoestande te beskryf en EKF’s om die landmerktoestande af te skat. FastSLAM 2 gebruik ’n verfyningsproses om die grootte van die voorstelverspreiding te verminder en dus die aantal partikels wat vir akkurate SLAM benodig word, te verminder.

Ons toets die drie SLAM-algoritmes deeglik in ’n simulasie-omgewing en vind dat al drie onder die regte omstandighede akkurate resultate kan behaal. EKF SLAM is egter baie sensitief vir foutiewe landmerkooreenkomste. FastSLAM is meer bestand daarteen, maar kan nie die sesdimensionele verspreiding wat vir 3D SLAM vereis word, beskryf nie. FastSLAM 2 bied ’n goeie kompromie tussen effektiwiteit en akkuraatheid, en presteer oor die algemeen goed.

Ons toets die hele stelsel met werklike data om dit te evalueer, en vind dat ons uitskieterherkenningsalgoritme baie effektief is en die akkuraatheid van die SLAM-stelsels beduidend verbeter. Ons vergelyk resultate van die drie SLAM-stelsels met onafhanklike DGPS-data, wat as korrek beskou kan word, en behaal akkuraatheid wat vergelykbaar is met ander toonaangewende stelsels.

Ons resultate lei tot die gevolgtrekking dat stereo-visie ’n lewensvatbare sensor vir SLAM is.

# Contents

# Nomenclature

### Abbreviations and Acronyms

| | |
|---|---|
| 2D | Two-dimensional |
| 3D | Three-dimensional |
| CCD | Charge-coupled device |
| DGPS | Differential global positioning system |
| DoG | Difference of Gaussians |
| EIF | Extended information filter |
| EKF | Extended Kalman filter |
| FAST | Features from accelerated segment test |
| GPS | Global positioning system |
| IF | Information filter |
| KF | Kalman filter |
| RANSAC | Random sample consensus |
| SEIF | Sparse extended information filter |
| SIFT | Scale-invariant feature transform |
| SLAM | Simultaneous localization and mapping |
| SURF | Speeded-up robust features |
| SVD | Singular value decomposition |

### Notation

| | |
|---|---|
| $x$ | Scalar |
| $X$ | Set or axis |
| $\mathbf{x}$ | Column vector |
| $\mathbf{X}$ | Matrix |
| $\mathbf{0}$ | Zero column vector |
| $\mathbf{O}_{n\times m}$ | Zero matrix of size $n\times m$ |
| $\mathbf{J}_f$ | Jacobian of $f$ |
| $x_{i,t}^{[k]}$ | $i$-th variable corresponding to particle $k$ at time step $t$ |

**Coordinate systems**

| | |
|---|---|
| $X_{\text{im}}$, $Y_{\text{im}}$ | Image axes |
| $X_c$, $Y_c$ , $Z_c$ | Camera axes |
| $X_r$, $Y_r$ , $Z_r$ | Robot axes |
| $X_w$, $Y_w$ , $Z_w$ | World axes |
| $\mathbf{x}_{\text{im}} = \begin{bmatrix} x_{\text{im}} & y_{\text{im}} \end{bmatrix}^T$ | Point in image coordinates |
| $\mathbf{x}_L = \begin{bmatrix} x_L & y_L \end{bmatrix}^T$ | Point in image coordinates in left image |
| $\mathbf{x}_R = \begin{bmatrix} x_R & y_R \end{bmatrix}^T$ | Point in image coordinates in right image |
| $\mathbf{x}_c = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix}^T$ | Point in camera coordinates |
| $\mathbf{x}_r = \begin{bmatrix} x_r & y_r & z_r \end{bmatrix}^T$ | Point in robot coordinates |
| $\mathbf{x}_w = \begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$ | Point in world coordinates |

**Variables**

| | |
|---|---|
| $b$ | Baseline |
| $f$ | Focal length |
| $p_x$ | $x$-offset of principal point |
| $p_y$ | $y$-offset of principal point |
| $\mathbf{c}_L$ | Location vector of left camera centre |
| $\mathbf{c}_R$ | Location vector of right camera centre |
| $\mathbf{P}$ | Camera matrix |
| $\mathbf{K}$ | Calibration matrix |
| $\mathbf{R}$ | Rotation matrix |
| $D_p$ | Distortion parameters |
| $\mathbf{F}$ | Fundamental matrix |
| $\mathbf{z}_t$ | Measurement at time $t$ |
| $\mathbf{z}_{i,t}$ | Landmark measurement $i$ in $\mathbf{z}_t$ |
| $\mathbf{x}_t$ | Robot state vector at time $t$ |
| $x_t$ | Robot location state on $X$ axis |
| $y_t$ | Robot location state on $Y$ axis |
| $z_t$ | Robot location state on $Z$ axis |
| $\theta_t$ | Robot orientation state around the $X$ axis (roll angle) |
| $\phi_t$ | Robot orientation state around the $Y$ axis (pitch angle) |
| $\psi_t$ | Robot orientation state around the $Z$ axis (yaw angle) |
| $\mathbf{m}_t$ | Map at time $t$ |
| $\mathbf{m}_{j,t}$ | Landmark $j$ in $\mathbf{m}_t$ |

| | |
|---|---|
| $\mathbf{c}_t$ | Correspondence vector |
| $\mathbf{u}_t$ | Robot control input at time $t$ |
| $\mathbf{g}$ | Motion function |
| $\mathbf{h}$ | Measurement function |
| $\mathbf{G}$ | Jacobian of motion function |
| $\mathbf{H}$ | Jacobian of measurement function |
| $\mathbf{S}_t$ | Motion noise covariance (process noise) |
| $\mathbf{Q}_i$ | Noise covariance of measurement $i$ |
| $\boldsymbol{\mu}_t$ | EKF mean |
| $\boldsymbol{\Sigma}_t$ | EKF covariance |
| $Y_t$ | Set of particles |
| $w$ | Particle weight |

# Chapter 1

# Introduction

## 1.1 Background and motivation

With recent advances in computing power and sensor technology the development of a fully autonomous vehicle has become a real possibility. The benefits of having an autonomous vehicle are virtually endless in areas such as military reconnaissance, mining, underwater exploration, space exploration, domestic work and many other tasks that are either too dangerous or too tedious for humans to perform.

Simultaneous localization and mapping (SLAM) is a rapidly growing part of the autonomous navigation field. In short, SLAM attempts to solve the problem of building a map of an unknown environment, using the sensors on a mobile robot, while estimating the location of the robot using the partially built map. This presents a chicken-and-egg situation: an accurate map is necessary for localization, and accurate localization is necessary to build a map. The inter-dependency between the estimates of the robot location and the map of the environment, coupled with the inherent uncertainties in measurements and robot motion, makes SLAM an interesting and challenging research problem.

In order to develop a truly autonomous robot we believe it is vital to implement SLAM. Accurate localization is very important for processes such as path planning, and conflict detection and resolution. An accurate map of the environment also enables a robot to navigate through complex or cluttered environments.

The SLAM problem is considered to have been solved on a theoretical level. Several algorithms have been created that build a probabilistic map by filling it with landmarks as they are observed by the robot's sensors. Using the locations of landmarks in the map and new landmark measurements these algorithms can successfully estimate the robot's location relative to these landmarks, while simultaneously refining the location estimates of the landmarks. Figure 1.1 illustrates the basic process followed by a typical SLAM system. At the first time step the robot observes some landmarks and adds them to the map (which is usually built relative to the robot's starting pose). At the next time step the robot observes two of the landmarks in the map and two new landmarks. Using the landmarks in the map it can estimate its own position and the locations of the new landmarks in the map. This process is repeated at every subsequent time step.

Successful implementation of SLAM has, however, given rise to some issues that still need to be resolved, the most prominent of these being sensor related. If landmarks cannot be identified accurately and tracked over time, SLAM will be practically impossible. This problem is often referred to as the landmark correspondence problem.

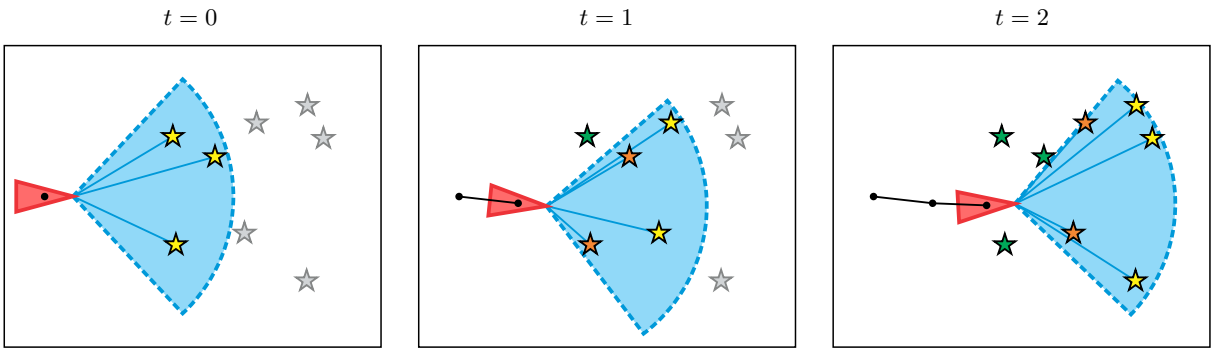



**Figure 1.1** – The basic process of SLAM. The red triangle represents a robot with its field of view in blue. Landmarks that are observed for the first time are depicted as yellow stars. Landmarks in the map are depicted as green stars if they are not being observed, or orange stars if they are being observed and used to refine the location estimate of the robot and the other landmarks.

Vision systems have increased in popularity as sensors for mobile robotics in recent years. Cameras are generally not only much cheaper than alternative sensors such as laser range finders or radar systems, but they also capture more information per sample. It may, however, be difficult to extract the information and convert it into a usable form. Fortunately the body of research on computer vision is very large and well documented.

Stereo vision refers to the use of two cameras that capture images of a scene from different view points (in our case, at the same time). The advantage of using two cameras is that 3D information can be extracted from the two images if corresponding points in the images can be found.

For these reasons the use of stereo vision as a sensor for SLAM has become an attractive research problem, and is also the focus of our research.

## 1.2 Problem statement and strategy

The main aim of this study is to establish whether stereo vision can be used effectively as a sensor for SLAM. In order to achieve this objective we aim to develop a complete SLAM system with stereo vision as the only sensor, and test it on real world data. This section describes the strategy we follow and how the execution of this strategy is presented in the thesis.

Through our literature study, given in Chapter 2, we establish a basic structure that most SLAM systems follow. This structure, upon which we base our system, divides the problem into two main sections: sensor and SLAM. The sensor section involves the conversion of raw sensor data into usable landmarks and the identification of correspondences between these new landmarks and landmarks in the map that have been seen before. The SLAM section uses these landmark measurements and correspondences to estimate the pose of the robot and the locations of the landmarks in an optimal fashion. One can argue that there is also a third part to this structure, namely an outlier detection algorithm, although it is often integrated into one of the other two parts. The outlier detection algorithm performs the task of identifying incorrect landmark correspondences.

Before we discuss the detection of landmarks we investigate the geometry of stereo cameras in order to establish a mathematical relationship between the scene observed by the cameras and the images they produce. This knowledge can be used to triangulate stereo image features pairs to 3D landmarks. Stereo camera geometry and calibration methods are described in Chapter 3.

In order to identify landmarks we explore the use of image feature detection algorithms. These algorithms are designed to find salient features in images that remain stable despite changes in the point of view, and to provide a descriptor for each feature. By using the descriptors we match features between stereo image pairs and calculate a set of 3D landmarks at every time step. In order to be able to use landmark measurements in a probabilistic fashion we also provide a method to quantify uncertainty for each of the measured landmarks. These techniques are described in the first part of Chapter 4.

The process of establishing whether a measured landmark should be classified as a new landmark or, if it has been observed before, to which landmark in the map it corresponds is in our view the most important and difficult part of the entire system. Through tests we see that poor landmark matching can have a profoundly negative effect on accuracy. We handle this crucial process as a separate part of the system and design it in such a way that it can be used with different feature detectors or SLAM algorithms. A large part of Chapter 4 is dedicated to landmark matching and introduces a novel method for identifying landmark mismatches.

We explore the use of three different SLAM algorithms firstly on a conceptual level in Chapter 5, and then through simulation in Chapter 6. The algorithms we choose are EKF SLAM, FastSLAM and FastSLAM 2. EKF SLAM is based on the extended Kalman filter (EKF) and estimates the robot pose and landmark locations by describing the entire system as a multivariate Gaussian distribution, with robot and landmark states. The other two algorithms are based on the Rao-Blackwellized particle filter and use both particles and Gaussian distributions to describe the system. We provide a detailed description of all three with the advantages and disadvantages of each.

In an attempt to answer our research question we evaluate the complete system on real world datasets. Chapter 7 gives details of our mobile robot and camera rig as well as the environment in which we captured the datasets. This is followed by a description of the practical results we obtained and a few interesting observations.

The final chapter summarizes our findings and conclusions. We also provide some ideas for future work.

# Chapter 2

# Related work

In this chapter we discuss some existing literature related to stereo vision-based SLAM. From the literature we establish a common structure for developing a solution to the problem, and we proceed to explore each of the main components of this structure.

## 2.1 Vision-based SLAM

Many researchers have considered the problem of performing SLAM with stereo vision as a sensor. Notable examples include the work of Ahn et al. [2], Agrawal and Konolige [1], Civera et al. [9], Dubbelman et al. [11], Grisetti et al. [15], Paz et al. [28], Se et al. [31], Vedaldi et al. [39] and Zhang and Koseka [41]. From these and other sources we observe a basic structure that almost all vision-based SLAM systems follow (at least to a large extent).

The structure, shown in Figure 2.1, is executed at every time step. First, image features are extracted from the incoming pair of stereo images. The feature detectors are designed to find salient points in an image and to describe each feature with a descriptor vector. Using these descriptors, image features are matched between the stereo images. Every feature pair can then be triangulated to a 3D landmark. The measured landmarks are matched to the landmarks in the map to establish whether they have been observed at previous time steps. If this matching process indicates that a landmark has been observed before, that information is stored in a correspondence vector. Because of unavoidable matching errors resulting from the sometimes ambiguous nature of image features,

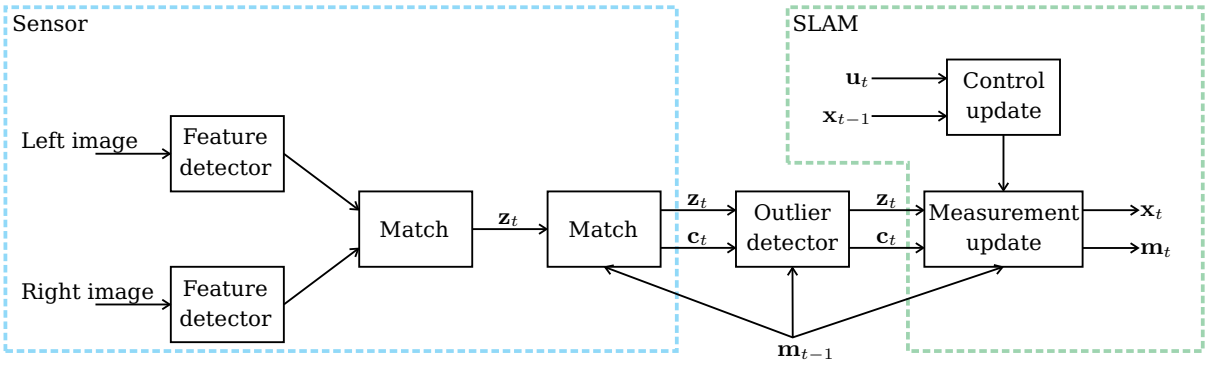



**Figure 2.1** – The structure of a typical stereo vision-based SLAM system, as executed at each time step. The landmark measurement is denoted with $\mathbf{z}_t$, the landmark-based map with $\mathbf{m}_t$, the correspondence vector relating measured landmarks to the map with $\mathbf{c}_t$, the control input with $\mathbf{u}_t$ and the robot pose with $\mathbf{x}_t$.

the matches are subjected to an outlier detection algorithm that is often built into the SLAM component. Confirmed matches are then used to perform SLAM. Following the basic Bayesian estimation structure, the SLAM algorithm is usually divided into a prediction or control update part and a measurement update part.

The main structure can be separated into a sensor component and a SLAM component. The third component, namely outlier detection, can be included in either one of the other two or, kept as a separate component as in our system.

## 2.2 Feature detectors

In order to perform SLAM we need to identify landmarks in the environment. Most stereo vision-based SLAM systems use some kind of feature detector to find points in stereo images. By matching these points across a synchronized stereo image pair we can extract 3D point landmarks. There are, however, a large number of feature detectors available to choose from. We begin our search by specifying what we look for in a feature detector, and then consider a few algorithms that are available.

Based on work by Zheng et al. [42] and Tuytelaars and Mikolajczyk [36] the following measures can be used to evaluate feature detectors.

- **Repeatability:** The same features should be identified in images taken from different viewpoints.
- **Distinctiveness:** The underlying pattern of a feature should contain a high amount of variation to enable a unique description of each feature.
- **Robustness to noise:** The addition of image noise should not have a large influence on the performance of the feature detector.
- **Accuracy:** Features should be accurately localized in the image.
- **Efficiency:** Execution time of the feature detector should be as low as possible.

The first feature detector to be used with SLAM was probably Harris corners [16]. The algorithm detects vertical and horizontal edges by using a Sobel operator. To reduce noise, the edges are filtered with a two-dimensional Gaussian. The resulting edge components are then combined to form an energy map in which peaks indicate corners. Harris corners are historically significant but, due to low accuracy and repeatability, not really relevant to the field anymore.

In order to achieve better repeatability, Lowe created the scale-invariant feature transform (SIFT) [21]. This algorithm uses differences of Gaussians in a scale space framework to detect edges in images at different scales. Peaks and valleys on these edges are then identified as potential features. These features are localized up to subpixel accuracy using a Taylor series approximation of the scale space function. In-plane rotation invariance is achieved by assigning a dominant orientation to each feature. A descriptor is calculated for each feature using the image gradients of the surrounding pixels. Since the descriptor is calculated relative to the dominant orientation of each feature, and in scale space, it is invariant to scale and in-plane rotation. Steps are also taken to achieve a degree of invariance to illumination changes.

SIFT is a very accurate feature detector, and has become a benchmark against which others are often compared, but it is generally slow to execute. To overcome this limitation, Bay et al. [3]

developed speeded-up robust features (SURF). SURF is in many ways an approximation of SIFT. The main difference is that, instead of using differences of Gaussians, SURF uses Hessian responses which can be calculated very efficiently using integral images.

In recent years a trend in image feature detection has been to employ machine learning approaches. Rosten and Drummond [30] proposed a method of training a classifier based on how a feature should behave. They call this method features from accelerated segment test (FAST). These features can be adapted for specific applications by optimizing the classifier for different purposes like repeatability (FAST-ER) or speed (FAST-n).

We choose to investigate the use of SIFT and SURF because of their performance in terms of repeatability and accurate sub-pixel localization, two properties that are very important in a SLAM system.

## 2.3 SLAM algorithms

Solving the problem of simultaneously estimating the pose of a mobile robot and the locations of landmarks observed by the robot is seen as one of the great achievements of the robotics research community. This section gives a brief description of some of the available algorithms. We base our review of the SLAM algorithms on the work of Thrun et al. [34] and Durrant and Bailey [12] who published comprehensive and detailed descriptions of the algorithms.

There are two main branches to the SLAM problem: online SLAM and the full SLAM problem. Online SLAM entails estimating the robot pose and the map at every time step, i.e. estimating the posterior

$$p(\mathbf{x}_t, \mathbf{m} \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}), \tag{2.3.1}$$

with $\mathbf{x}_t$ the pose of the robot, $\mathbf{m}$ a map containing landmarks in the environment, and $\mathbf{z}_{1:t}$ and $\mathbf{u}_{1:t}$ respectively the measurements made by the robot and the control inputs given to the robot from the first time step up to the current time step. Figure 2.2 depicts a graphical representation of the problem. In the full SLAM problem the posterior over the entire path, $\mathbf{x}_{1:t}$, must be estimated.

When faced with such a recursive estimation problem, the Kalman filter or its extension for nonlinear systems, the extended Kalman filter (EKF), is usually the first algorithm that is considered and it was no different for SLAM. Smith et al. [33] established a statistical basis for describing the geometric relationship between landmarks, and from this Leonard and Durrant-Whyte [20] created the first version of EKF SLAM.

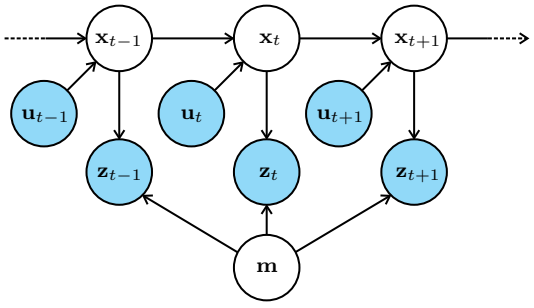



**Figure 2.2** – A graphical model of the SLAM problem. At every time step we want to estimate both the map $\mathbf{m}$ and the robot's state vector $\mathbf{x}_t$, given the previous state $\mathbf{x}_{t-1}$, a measurement $\mathbf{z}_t$ and a control input $\mathbf{u}_t$.

Like any other EKF (detailed in Appendix A.1), EKF SLAM describes the system states with a single multivariate Gaussian distribution. This is achieved by building a large state vector and covariance matrix corresponding to the robot and landmark states. Normal EKF equations can then be used at every time step to update the system, first with the control input to the robot and then with incoming landmark measurements.

Similar to the EKF, the information filter (IF) also uses a Gaussian distribution to describe the system states. Instead of using the moments (a mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$) to represent Gaussian distributions, the IF uses the canonical parameterization which comprises an information matrix $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$ and an information vector $\boldsymbol{\xi} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$. SLAM can be performed with the nonlinear extension of the IF, which is termed the extended information filter (EIF) [35]. The EIF uses exactly the same process as the EKF to linearize the system, by transforming the Gaussian distribution to its moment representation.

Because of this transformation of the system representation the EIF will be less efficient than the EKF. The benefit of using the EIF becomes apparent when a typical information matrix is considered. Figure 2.3 depicts a covariance matrix obtained through a simple SLAM simulation and its corresponding information matrix. We observe that the information matrix is fairly sparse when compared to the covariance matrix. This property of the SLAM information matrix implies that an extension of the EIF can be used, namely the sparse extended information filter (SEIF). With SEIF SLAM updates can be performed in constant time, and they are independent of the size of the map. For this reason SEIF SLAM can be considerably more efficient than EKF SLAM [34, p. 385].

EKF SLAM and SEIF SLAM use Gaussian distributions to describe the uncertainty in the system. An alternative to this is to use a Monte Carlo method that describes a distribution with a set of sample points. This leads us naturally to the particle filter (explained in Appendix A.2) but, because of the high dimensionality of the problem, we cannot use it directly. Instead, the Rao-Blackwellized particle filter is used. This adaptation of the particle filter uses particles to estimate the robot states and, for every particle, an EKF for every landmark. The resulting algorithm is called FastSLAM [25].

An extension to FastSLAM was developed to decrease the number of particles needed. This extension, called FastSLAM 2, uses equations akin to those of the Kalman filter to refine the distribution from which particles are drawn, thereby greatly decreasing the uncertainty in the proposal distribution [26]. Because of the smaller uncertainty fewer particles are needed.

The methods discussed so far (and that we choose to investigate further) can be classified as filtering approaches. Alternatively, smoothing approaches estimate the entire route of the robot with all the

(a) covariance matrix (b) information matrix

**Figure 2.3** – A graphic representation of the covariance matrix and corresponding information matrix from a typical SLAM simulation (white indicates zero). Note that the information matrix is fairly sparse when compared to the covariance matrix.

measurements in an effort to solve the full SLAM problem. One such method is called GraphSLAM [14]. GraphSLAM constructs a graph whose nodes correspond to the poses of the robot at different time steps, and edges represent constraints obtained from measurements and the control inputs to the robot. The route and map estimates can then be obtained by optimizing the spatial configuration of the nodes over the constraints imposed by the graph.

We focus on online SLAM, since it offers the possibility of being implemented in real-time applications (a possible focus of our future research). From the online algorithms, we choose to investigate the use of EKF SLAM and the two FastSLAM algorithms, as they encompass the two fundamental approaches of representing uncertainty: as a closed form function and as a Monte Carlo approximation. These algorithms are also without a doubt the most popular.

## 2.4 Outlier detection

Since feature matching is prone to errors, several researchers have focussed on outlier removal in vision-based SLAM. Most of the approaches rely on a random sample consensus (RANSAC) framework. RANSAC [13] is a method for finding data points in a set that do not conform, according to some consensus measure, to a specified model.

The fundamental matrix describes the relationship between points in two images of the same scene [17]. By using it as a model for RANSAC, outliers can be detected effectively between different images. Since we want to compare 3D landmarks rather than 2D image coordinates, it is not ideal for SLAM. It can, however, be used to detect outliers in consecutive frames. Agrawal and Konolige [1] use the 3D locations of landmarks in a RANSAC framework. We believe this approach to be somewhat suboptimal as a measurement noise model is not taken into consideration in their approach.

Visual odometry is in many ways comparable to SLAM. Nister et al. [27], for example, also use image features to estimate the route of a mobile robot. They propose the camera motion as a model for RANSAC and achieve fairly accurate localization.

Vedaldi et al. [39] and Civera et al. [9] both use a measurement model with the update step of an EKF in a RANSAC framework to find outliers. A limitation of these methods is that they require the EKF for state estimation, and therefore cannot be used with any other SLAM system.

Dubbelman et al. [11] propose a method that uses the Bhattacharyya distance in a RANSAC framework. They compare this method with an expectation maximization approach. From their work we deduce that the use of a probabilistic measure in a RANSAC framework can be very effective.

Using this literature review as a basis, we proceed by investigating each part of the vision-based SLAM structure in more detail, starting with the geometry of a stereo camera sensor that can be applied to extract 3D information from a pair of images.

# Chapter 3

# Stereo camera geometry and calibration

Before we can extract information from images, we need a formal mathematical description of how light travelling from objects is captured by a camera in the form of an image. We can then use these equations to reproject image information and estimate the location of the object relative to the camera. The topics covered in this chapter form the basis of almost all single and stereo vision systems in robotics. They have been well developed and documented by many researchers, such as Hartley and Zisserman [17] upon whose work this chapter is largely based.

The chapter starts with a derivation of the pinhole model for a single camera. We then describe a method to calculate the camera model parameters. We also consider the nonlinear effect on images caused by refraction as light travels trough the lens of a camera. The geometric relationship between two cameras is discussed as well as a method to transform image data in order to simplify the search for correspondences between two images. This transformation also simplifies the triangulation of two points in image coordinates to a 3D point.

## 3.1 Single camera geometry and calibration

We begin the discussion by defining a model for a single camera and describing a practical method for estimating the model parameters for a specific camera.

### 3.1.1 Camera model

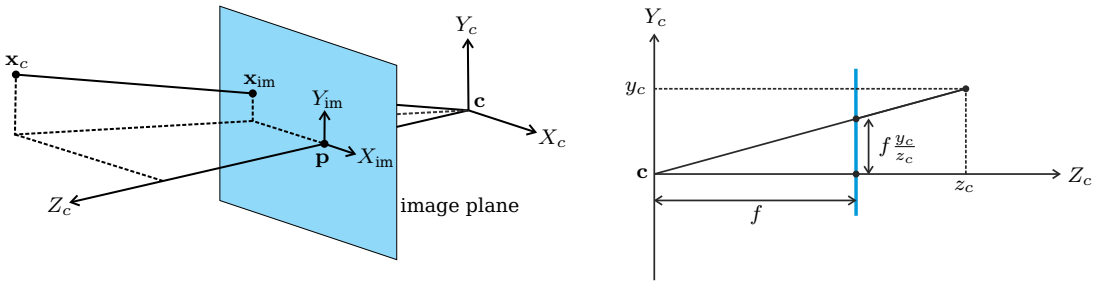
In order to describe the mathematical relationship between 3D points and their projections onto a 2D image plane, we use the pinhole camera model [17, p. 153]. Although the accuracy of this model depends on the camera being used, it has become the standard camera model in computer vision applications. Figure 3.1 depicts the basic geometry of the model. The camera axis system, denoted with subscript $c$, is defined to have its origin at the camera centre $\mathbf{c}$ with the $X_c$–$Y_c$ plane parallel to the image plane. The point where the $Z_c$ axis intersects the image plane, called the principal point, is usually close to the centre of the image. The distance between the camera centre and the image plane, which we call the focal length, is denoted by $f$. A 3D point $\mathbf{x}_c$ is captured at $\mathbf{x}_{\text{im}}$ on the image where the line connecting the point and the camera centre intersects the image plane.

Given that

$$\mathbf{x}_c = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix}^T, \tag{3.1.1}$$

we can use similar triangles to find the transformation from camera coordinates to image coordinates as

$$\mathbf{x}_{\text{im}} = \begin{bmatrix} f\frac{x_c}{z_c} + p_x & f\frac{y_c}{z_c} + p_y & f \end{bmatrix}^T. \tag{3.1.2}$$




(a) basic pinhole camera model (b) $Y_c$–$Z_c$ plane of pinhole camera model

**Figure 3.1** – The projection of a 3D point onto the image plane using the pinhole camera model.

Note that we include an offset to the image coordinate system with $p_x$ and $p_y$. This offset is necessary when the principal point is not at the exact centre of the image plane. It is also convenient to use this offset to move the origin of the image coordinate system (enabling us to have it coincide with matrix indexing, for example).

Formulating the nonlinear transformation in Equation 3.1.2 mathematically is simplified through the use of homogeneous coordinates. When using homogeneous coordinates to describe a 2D point, we use three parameters, so instead of $[x_{\text{im}} \quad y_{\text{im}}]^T$ we have $[u_{\text{im}} \quad v_{\text{im}} \quad w_{\text{im}}]^T$, with $x_{\text{im}} = \frac{u_{\text{im}}}{w_{\text{im}}}$ and $y_{\text{im}} = \frac{v_{\text{im}}}{w_{\text{im}}}$. Two homogeneous vectors are equivalent if they are equal up to scale, or

$$\begin{bmatrix} u_{\text{im}} \\ v_{\text{im}} \\ w_{\text{im}} \end{bmatrix} = \begin{bmatrix} \gamma u_{\text{im}} \\ \gamma v_{\text{im}} \\ \gamma w_{\text{im}} \end{bmatrix} \tag{3.1.3}$$

with $\gamma$ any nonzero scalar [17, p. 27].

The transformation in Equation 3.1.2 then becomes

$$\begin{bmatrix} u_{\text{im}} \\ v_{\text{im}} \\ w_{\text{im}} \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{3.1.4}$$

or, more concisely,

$$\begin{bmatrix} u_{\text{im}} \\ v_{\text{im}} \\ w_{\text{im}} \end{bmatrix} = \mathbf{P} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}, \tag{3.1.5}$$

where

$$\mathbf{P} = \mathbf{K}[\,\mathbf{I}\,|\,\mathbf{0}\,] \qquad \text{and} \qquad \mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.1.6}$$

The $3\times4$ matrix $\mathbf{P}$ is called the camera matrix, and $\mathbf{K}$ is called the calibration matrix (or sometimes the intrinsic parameter matrix).

In addition this model may account for the possibility of rectangular pixels, in other words different scales on the $X_{\text{im}}$ and $Y_{\text{im}}$ axes of the image. We modify the calibration matrix by multiplying it with

a scaling matrix, so that

$$\mathbf{K} = \begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.1.7}$$

with $m_x$ and $m_y$ the scaling factors. We now have a new calibration matrix $\mathbf{K}$ with $\alpha_x = fm_x$, $\alpha_y = fm_y$, $x_0 = p_x m_x$ and $y_0 = p_y m_y$.

We add one final parameter $s$ to generalize the camera model further, to allow for slightly skew pixels (parallelograms instead of rectangles). The generalized calibration matrix is

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.1.8}$$

Next we define an additional axis system, the robot axes, denoted with subscript $r$ (the reason for this name will become apparent later on). The camera will have some 3D location and orientation in this new coordinate system. To incorporate this extension in our model, we rotate and translate the point $\mathbf{x}_c$ in the camera coordinate system. In Euclidean coordinates this will give us

$$\mathbf{x}_c = \mathbf{R}(\mathbf{x}_r - \mathbf{c}), \tag{3.1.9}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{c}$ is the position of the camera centre in Euclidean robot coordinates. In homogeneous coordinates we have

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{Rc} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}. \tag{3.1.10}$$

This transformation gives us a new camera matrix,

$$\mathbf{P} = \mathbf{KR}\,[\,\mathbf{I}\,|\,-\mathbf{c}\,]. \tag{3.1.11}$$

This camera model has 11 degrees of freedom: 6 for the location and orientation of the camera in robot coordinates (also called the extrinsic parameters) and 5 for the internal specifications of the camera (the intrinsic parameters). We are now able to describe the projection of a point in front of the camera onto the image plane with the following simple equation in homogeneous coordinates:

$$\begin{bmatrix} x_{\text{im}} \\ y_{\text{im}} \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}. \tag{3.1.12}$$

### 3.1.2 Calibration

Now that we have a model that relates 3D coordinates with 2D image coordinates, the following question arises: how do we find the matrix $\mathbf{P}$ associated with a specific camera? As it turns out, $\mathbf{P}$ can be estimated if we have the 2D image coordinates of several points as well as their corresponding 3D robot coordinates [17, p. 178]. Suppose we have a number of points

$$\mathbf{x}_{i,\text{im}} = \begin{bmatrix} x_{i,\text{im}} & y_{i,\text{im}} & 1 \end{bmatrix}^T \tag{3.1.13}$$

in image coordinates, corresponding with points

$$\mathbf{x}_{i,r} = \begin{bmatrix} x_{i,r} & y_{i,r} & z_{i,r} & 1 \end{bmatrix}^T \tag{3.1.14}$$

in homogeneous robot coordinates, for $i = 1, \ldots, n$. We write $\mathbf{P}$ in term of its rows as

$$\mathbf{P} = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix}, \tag{3.1.15}$$

and from Equation 3.1.12 we have

$$\mathbf{x}_{i,\text{im}} = \mathbf{P}\mathbf{x}_{i,r} = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix} \mathbf{x}_{i,r} = \begin{bmatrix} \mathbf{r}_1^T \mathbf{x}_{i,r} \\ \mathbf{r}_2^T \mathbf{x}_{i,r} \\ \mathbf{r}_2^T \mathbf{x}_{i,r} \end{bmatrix}. \tag{3.1.16}$$

Since we are working in homogeneous coordinates, $\mathbf{x}_{i,\text{im}}$ and $\mathbf{P}\mathbf{x}_{i,r}$ are not strictly equal but can be viewed as vectors pointing in the same direction, possibly with different magnitudes. To get around this problem, we express Equation 3.1.16 in the form of a cross product as $\mathbf{x}_{i,\text{im}} \times \mathbf{P}\mathbf{x}_{i,r} = \mathbf{0}$. This form gives us a strict equality with a linear solution. In vector cross product form Equation 3.1.16 is given by

$$\begin{bmatrix} \mathbf{0}^T & -\mathbf{x}_{i,r}^T & y_{i,\text{im}}\mathbf{x}_{i,r}^T \\ \mathbf{x}_{i,r}^T & \mathbf{0}^T & -x_{i,\text{im}}\mathbf{x}_{i,r}^T \\ -y_{i,\text{im}}\mathbf{x}_{i,r}^T & x_{i,\text{im}}\mathbf{x}_{i,r}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \mathbf{0}. \tag{3.1.17}$$

This is of the form $\mathbf{A}_i\mathbf{r} = \mathbf{0}$ with $\mathbf{r}$ unknown. Since only two of the rows in $\mathbf{A}_i$ are linearly independent the system reduces to

$$\begin{bmatrix} \mathbf{0}^T & -\mathbf{x}_{i,c}^T & y_{i,\text{im}}\mathbf{x}_{i,c}^T \\ \mathbf{x}_{i,c}^T & \mathbf{0}^T & -x_{i,\text{im}}\mathbf{x}_{i,c}^T \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \mathbf{0}. \tag{3.1.18}$$

Thus for every point correspondence we obtain two equations. Since the camera matrix has 11 degrees of freedom we need at least $5\frac{1}{2}$ point correspondences to solve for $\mathbf{P}$.

In practice an image of a calibration object with known 3D dimensions can be used to obtain the set of points correspondences. We use the Camera Calibration Toolbox for MATLAB [5] to perform our calibration, with several images of a checkerboard with known dimensions. Since the relative 3D locations of the board are unknown, the toolbox also uses an optimization algorithm to minimize the reprojection error over the intrinsic camera parameters and the positions of the board.

### 3.1.3 Radial distortion

Up to now we have assumed that light travels in a perfectly straight line. However, when light travels through the lens of a camera, it will usually bend to some degree because of refraction. This phenomenon is called lens distortion and, in order to keep the camera model valid, we have to compensate for it. In order to do this we need a way to model the distortion [17, p. 189].

The amount of distortion is normally modelled to be proportional to the distance from what is called the centre of distortion (usually close to the principal point) given by $[x_g \quad y_g]^T$. For this reason lens distortion is often referred to as radial distortion. If the distorted point on the image is given by $[x_d \quad y_d]^T$ and the corrected point is given by $[x_{\text{im}} \quad y_{\text{im}}]^T$, the distortion can be described with a function $d(r)$, where $r = \sqrt{(x_d - x_g)^2 + (y_d - y_g)^2}$ is the distance from the centre of distortion. This description is expressed as

$$\begin{bmatrix} x_{\text{im}} \\ y_{\text{im}} \end{bmatrix} = \begin{bmatrix} x_g \\ y_g \end{bmatrix} + d(r) \begin{bmatrix} x_d - x_g \\ y_d - y_g \end{bmatrix}. \tag{3.1.19}$$

(a) with radial distortion

(b) without radial distortion

**Figure 3.2** – The removal of radial distortion.

Since the distortion function is unknown we describe it with a Taylor expansion as

$$d(r) = 1 + \kappa_1 r + \kappa_2 r^2 + \kappa_3 r^3 + \dots . \tag{3.1.20}$$

These distortion parameters $D_p = \{x_g, y_g, \kappa_1, \kappa_2, \kappa_3, \dots\}$ can be viewed as additional intrinsic parameters of the camera.

In practice sufficiently accurate results can be obtained by using a second or third order approximation of $d$. These parameters are obtained by identifying lines that should be straight on an image and then solving $D_p$ with some optimization algorithm. The lines can be identified by hand or automatically using a calibration object such as a checkerboard pattern. An example of the effect and removal of radial distortion is given in Figure 3.2.

With the camera matrix and distortion parameters we can describe the relationship between objects in the world and one camera. In order to enable 3D reconstruction we now continue to the geometry of two cameras.

## 3.2 Stereo camera calibration and geometry

With one camera we can find the 3D ray on which a point with known image coordinates lies. If we can find the image coordinates of a point in two images we can determine the 3D location where the rays cross. This section describes how we go about calibrating two cameras and triangulating image coordinates to 3D points. We also discuss a method for simplifying the search for corresponding image points called image rectification.

### 3.2.1 Calibration

In order to work with two cameras simultaneously, we have to be able to determine their geometric relationship. As it turns out, we already possess all the necessary tools.

When two cameras are synchronized to capture images at the same time, we can capture images of a calibration object. From these images we extract image coordinates with known 3D positions. When the individual cameras are calibrated with the same 3D coordinates, the calibration process gives us the spatial relationship between the two cameras, relative to the same coordinate frame, as well as the intrinsic parameters of each camera.

Some examples of calibration image pairs are shown in Figure 3.3. The points used for calibration are also shown, as identified by the corner detection algorithm of the calibration toolbox.

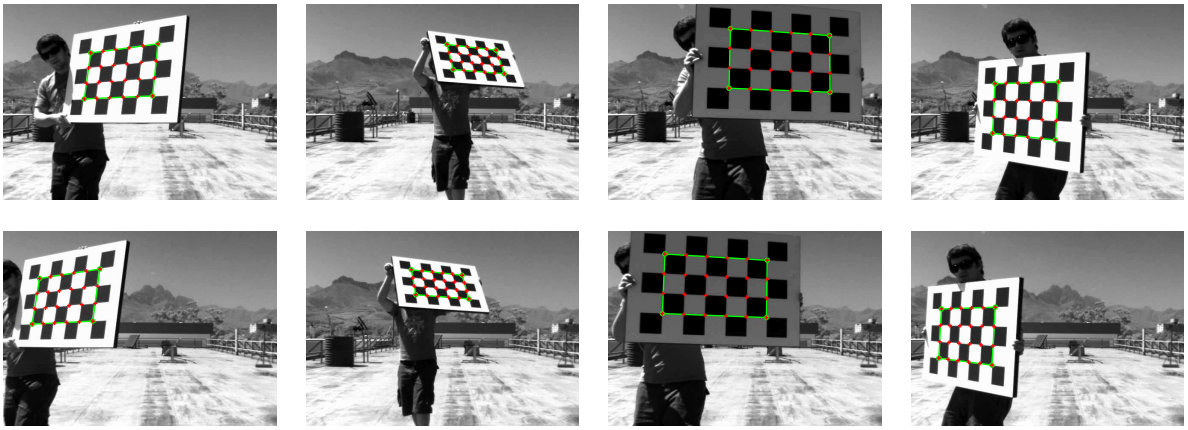


**Figure 3.3** – Examples of stereo calibration images. Images captured by the left camera are shown in the top row with corresponding images captured by the right camera in the bottom row.

### 3.2.2 Epipolar geometry

If we have identified a pair of image coordinates corresponding to the same 3D point, say $\mathbf{x}_L = [x_L \quad y_L]^T$ in one image and $\mathbf{x}_R = [x_R \quad y_R]^T$ in the other, we can find the 3D location of the point by solving for $[x_r \quad y_r \quad z_r]^T$ from

$$\begin{bmatrix} x_L \\ y_L \\ 1 \end{bmatrix} = \mathbf{K}_L \mathbf{R}_L \left[\, \mathbf{I} \,|\, -\mathbf{c}_L \,\right] \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x_R \\ y_R \\ 1 \end{bmatrix} = \mathbf{K}_R \mathbf{R}_R \left[\, \mathbf{I} \,|\, -\mathbf{c}_R \,\right] \begin{bmatrix} x_r \\ y_r \\ z_r \\ 1 \end{bmatrix}, \tag{3.2.1}$$

where a subscript $L$ denotes the left image and a subscript $R$ the right image. These equations can be written as cross products equal to zero, and solved by using the SVD. It is, however, useful to first discuss two-view geometry (or epipolar geometry) in more depth.

Figure 3.4 depicts two cameras with centres at $\mathbf{c}_L$ and $\mathbf{c}_R$. A 3D point $\mathbf{x}_r$ is projected onto the two image planes at $\mathbf{x}_L$ and $\mathbf{x}_R$. The line segment connecting the two camera centres is called the baseline, and we denote its length by $b$. The plane formed by the camera centres and $\mathbf{x}_r$ is called an epipolar plane. Suppose we know $\mathbf{x}_L$, but neither $\mathbf{x}_r$ nor $\mathbf{x}_R$. We know that $\mathbf{x}_r$ is on the line through $\mathbf{c}_L$ and $\mathbf{x}_L$, hence the corresponding $\mathbf{x}_R$ will fall on the line where the epipolar plane intersects the image plane. This line is called an epipolar line. This useful property of two-view geometry (called the epipolar constraint) limits the search space to a single line, as opposed to the entire image, when we attempt to identify image coordinate correspondences. The points of intersection of the baseline and the image planes are called epipoles. Note that all epipolar lines of an image will intersect at the epipole [17, p. 239].

By using properties of the epipoles and the epipolar lines it can be shown that there exists a $3\times3$ homogeneous matrix $\mathbf{F}$, called the fundamental matrix, that satisfies

$$\mathbf{x}_R^T \mathbf{F} \mathbf{x}_L = 0 \tag{3.2.2}$$

for any $\mathbf{x}_L$ and $\mathbf{x}_R$ corresponding to the same 3D point [17, p. 241]. This relationship encapsulates the above-mentioned epipolar constraint and will become useful in the next chapter. The fundamental matrix has a rank of two and seven degrees of freedom.

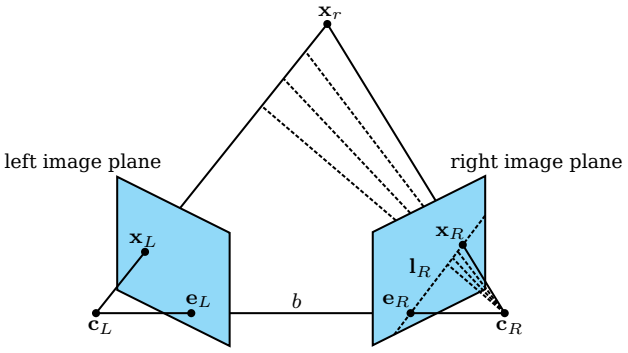



**Figure 3.4** – Epipolar geometry.

We now have a way of decreasing our search space for image coordinate pairs and, once they are found, we can transform them to 3D coordinates. Both of these processes can be greatly simplified by means of a procedure called image rectification.

### 3.2.3 Image rectification

The main purpose of image rectification is to transform the two images so that they have the same intrinsic parameters and are parallel to the baseline [17, p. 303]. The only difference in the camera matrices should be an offset along one of the main axes. Another way of describing this is that if we have the two camera matrices

$$\mathbf{P}_L = \mathbf{M}_L[\mathbf{I}\,|\,-\mathbf{c}_L] \qquad \text{and} \qquad \mathbf{P}_R = \mathbf{M}_R[\mathbf{I}\,|\,-\mathbf{c}_R], \tag{3.2.3}$$

with $\mathbf{M}_L = \mathbf{K}_L\mathbf{R}_L$ and $\mathbf{M}_R = \mathbf{K}_R\mathbf{R}_R$, we want to find a common $\mathbf{M} = \mathbf{KR}$ for the two cameras so that

$$\mathbf{P}'_L = \mathbf{M}[\mathbf{I}\,|\,-\mathbf{c}_L] \qquad \text{and} \qquad \mathbf{P}'_R = \mathbf{M}[\mathbf{I}\,|\,-\mathbf{c}_R]. \tag{3.2.4}$$

We begin by choosing

$$\mathbf{K} = \frac{1}{2}(\mathbf{K}_L + \mathbf{K}_R), \tag{3.2.5}$$

to minimize the distortion on the images brought on by the transformation.

For the sake of convenience we fix the robot coordinate system to the centre of the baseline as depicted in Figure 3.5 (b). We now want the images to be coplanar and therefore want the new $X_r$ axis to be perpendicular to the baseline. To achieve this we make the second column vector of $\mathbf{R}$, which corresponds to the new $Y_r$ axis, parallel to the baseline, that is

$$\mathbf{r}_2 = \frac{\mathbf{c}_R - \mathbf{c}_L}{||\mathbf{c}_R - \mathbf{c}_L||}. \tag{3.2.6}$$

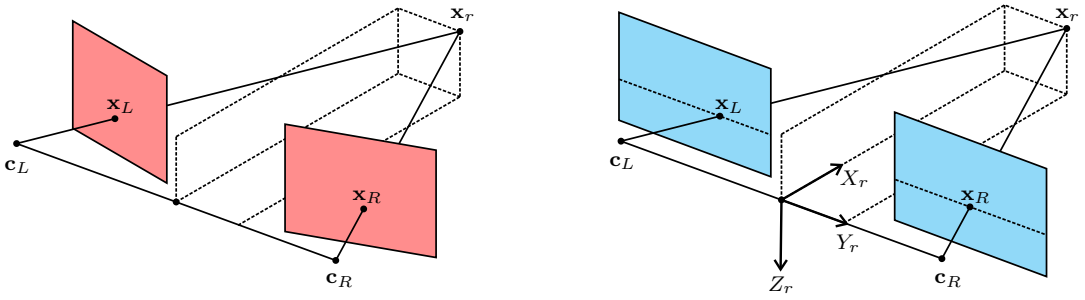
The new $Z_r$ axis has to be orthogonal to $\mathbf{r}_2$, but is otherwise unconstrained. In order to minimize distortion we choose

$$\mathbf{r}_3 = \frac{\mathbf{k} \times \mathbf{r}_2}{||\mathbf{k} \times \mathbf{r}_2||} \tag{3.2.7}$$

where $\mathbf{k}$ is the unit vector along the $Z_c$ axis of one of the cameras.

The final unit vector, corresponding to the new $X_r$ axis, has to be orthogonal to the other two, so

$$\mathbf{r}_1 = \mathbf{r}_2 \times \mathbf{r}_3. \tag{3.2.8}$$




(a) unrectified stereo pair (b) rectified stereo pair

**Figure 3.5** – Rectification of stereo image planes.

This gives us

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix}. \tag{3.2.9}$$

To transform the images we map every pixel according to

$$\mathbf{x}'_L = \mathbf{M}\mathbf{M}_L^{-1}\mathbf{x}_L \qquad \text{and} \qquad \mathbf{x}'_R = \mathbf{M}\mathbf{M}_R^{-1}\mathbf{x}_R. \tag{3.2.10}$$

This transformation back-projects all the points in the old image plane and then reprojects them, using the new parameters. In essence the original camera properties are simply removed and replaced with new ones.

This rectification maps the epipoles to infinity, which is exactly what we want. Since the epipoles are the points were epipolar lines intersect, the epipolar lines have been made parallel. With this transformation we can also ensure that all image coordinate correspondences must have the same vertical coordinates.

### 3.2.4 Triangulation

Image rectification also simplifies the process of triangulation considerably. Consider Figure 3.6, where the geometry of a rectified stereo camera pair is depicted from above and from the side. We have corresponding points on the two rectified image planes, respectively $[x_L \quad y]^T$ and $[x_R \quad y]^T$, and wish to calculate the coordinates of $\mathbf{x}_r$. When similar triangles are used,

$$\frac{f}{x_L} = \frac{x_r}{\frac{b}{2} + y_r} \qquad \text{and} \qquad \frac{f}{-x_R} = \frac{x_r}{\frac{b}{2} - y_r}, \tag{3.2.11}$$

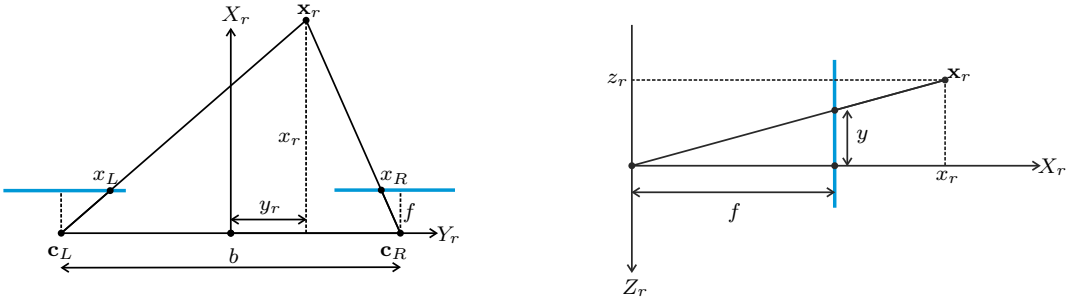
which give

$$x_r = \frac{fb}{x_L - x_R} \qquad \text{and} \qquad y_r = \frac{x_L b}{x_L - x_R}. \tag{3.2.12}$$

When Figure 3.6 (b) is used in a similar fashion, we derive that

$$z_r = \frac{yb}{x_L - x_R}. \tag{3.2.13}$$

If we add the offsets $p_x$ and $p_y$ to $x_L$, $x_R$ and $y$ we get




(a) $X_r$–$Y_r$ plane (b) $X_r$–$Z_r$ plane

**Figure 3.6** – Rectified stereo geometry viewed from above and the side.

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} \frac{fb}{x_L - x_R} \\ \frac{(x_L - p_x)b}{x_L - x_R} - \frac{b}{2} \\ \frac{(y - p_y)b}{x_L - x_R} \end{bmatrix}. \tag{3.2.14}$$

This illustrates another benefit of rectification: the entire geometry of the rectified system can now be described with only four parameters: $f$, $b$, $p_x$ and $p_y$.

We have now established a camera model and can use it with stereo geometry to calculate the 3D location of a point found in two images. The next chapter focuses on the problem of finding corresponding image coordinates in rectified images.

# Chapter 4

# Image landmarks

Before attempting SLAM, it is vital to have well defined landmarks that can be tracked in 3D over multiple time steps. In the previous chapter we saw that if we have image coordinates of a point in two calibrated images taken at the same time, we can find the 3D location of that point. The question we now ask is: how do we choose corresponding image coordinates at one time step to create a 3D landmark, and then find that landmark again in subsequent image pairs?

Our answer to this question is to use a feature detection and matching algorithm. We investigate two popular algorithms, namely the scale invariant feature transform (SIFT) and speeded-up robust features (SURF). These algorithms are each designed to find salient features in an image and build a unique descriptor for each feature. The descriptors can then be used to match features, in our case between the left and right images and then also over time. In this chapter we provide a detailed discussion on matching features with descriptors, with a number of additional steps to identify and eliminate incorrect matches.

In order to use features as landmarks for probabilistic SLAM we have to characterize each measured landmark with a noise model. We choose to use a Gaussian model, derived with the use of a first order Taylor approximation.

Despite our best efforts, matching with descriptors is prone to errors. To increase accuracy we subject putative matches from the initial matching phase to a novel variation of random sample consensus (RANSAC). We argue that our variation is more suited to the SLAM problem than the standard use of RANSAC with the fundamental matrix.

## 4.1 Feature detection and matching

The aim of a feature detection algorithm is to find salient, distinct points in an image. The idea is that the same features will be detected in different images of a particular scene despite changes in the position of the camera, i.e. scale change, rotation change, projective transformation, and change in illumination.

Once features have been detected, we need a way of distinguishing them from one another. We use descriptors that contain information about each feature and these descriptors must ideally be invariant to changes in the viewpoint of the camera.

Methods for feature detection and matching have been the focus of much research over the last 25 years. From the multitude of algorithms available we choose to focus on two popular ones: the scale invariant feature transform (SIFT), because of its accuracy, and speeded-up robust features (SURF) because of its speed.

### 4.1.1 SIFT

SIFT is arguably the most well-known algorithm to detect features and even now, more that 13 years after its original publication [21], remains a benchmark against which others are tested and often outperforms state-of-the art algorithms in terms of accuracy. The high accuracy of the algorithm does, however, come at the price of a high execution time.

The description we give of the algorithm is derived from a more recent publication [22] and we use the implementation provided with it (which is freely available online [23]) in all our tests. The algorithm can be divided into two parts: finding distinctive features, and calculating descriptors for those features.

The first step of the algorithm is to build scale space pyramids and, from them, difference of Gaussian (DoG) pyramids. When an object moves further away from a camera, the detail of the object in the image is reduced. This can by imitated by blurring the image to reduce detail, or simply by down-sampling the image. SIFT builds scale-space pyramids by doing both. An input image $I(x, y)$ is blurred by convolving it with a Gaussian kernel. A blurred image is therefore calculated as

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{4.1.1}$$

with

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}(x^2+y^2)}. \tag{4.1.2}$$

By increasing $\sigma$, different levels of scale change can be created. The DoG image will then be

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{4.1.3}$$

where $k$ is a constant. This blurring and DoG calculation are repeated on a down-sampled image. Figure 4.1 depicts two octaves (sample levels) that may result from this process.

Strong corners in an image will typically be manifested as local extrema in a DoG, as is evident from Figure 4.2, and stability with respect to scale changes is accomplished by identifying extrema across different DoGs. The initial search for features is therefore done by selecting extrema in the DoG pyramid, with an extremum defined as a pixel with the highest or lowest value compared to its 26 neighbours in the pyramid. Figure 4.1 clarifies.

The next step of the algorithm is to refine the location of the feature in the image. By using a Taylor series to approximate the scale space function $D(x, y, \sigma)$ as

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}, \tag{4.1.4}$$

we introduce the offset of the feature from the sample point $\mathbf{x} = [x \quad y \quad \sigma]^T$. The function $D$ and its derivatives are evaluated at the sample point. The location $\hat{\mathbf{x}}$ of a feature is determined by taking the derivative of Equation 4.1.4 with respect to $\mathbf{x}$ and setting it to zero, giving

$$\hat{\mathbf{x}} = -\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial D}{\partial \mathbf{x}}. \tag{4.1.5}$$

With this approximation the location of the feature is estimated to sub-pixel precision.

The initial set of features may contain many features that are poorly suited for tracking. Such points will typically have low contrast, making them sensitive to noise, or may be poorly localized on an edge. The function value at the feature, $D(\hat{\mathbf{x}})$, is used to reject features with low contrast. By substituting $\hat{\mathbf{x}}$ into Equation 4.1.4, this value can be obtained and compared to a threshold.

Calculating a difference of Gaussians can be viewed as a method to perform edge detection since a DoG will have a strong response along edges, as is illustrated in Figure 4.2. However, features on

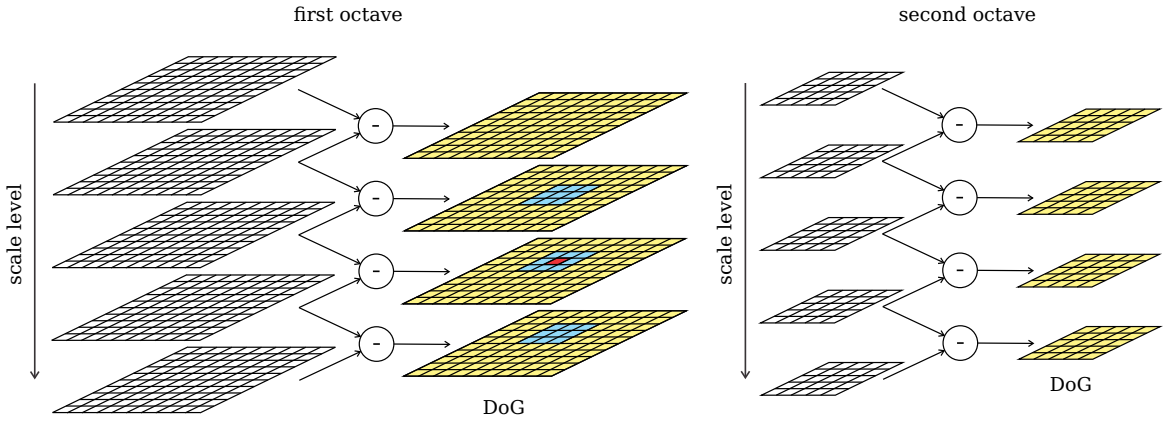



**Figure 4.1** – Scale space pyramids are built by repeatedly blurring an image with a Gaussian filter, from which a difference of Gaussian (DoG) pyramid can then be calculated. This process is repeated on a down-sampled image to imitate greater scale change. A feature which is an extremum when compared to its neighbours is selected. Here an extremum is shown in red, with its neighbours in blue.

edges are not particularly stable as a small amount of noise can cause them to move to a different position on the edge. An extremum in a DoG on an edge will have a much smaller curvature (a measure of how much a surface deviates from being flat) along the edge than in the perpendicular direction. These curvatures can be determined with the use of a 2×2 Hessian matrix,

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}, \tag{4.1.6}$$

where the entries are the second order derivatives of $D$. These derivatives can be estimated by using differences between neighbouring pixels. The eigenvalues of the Hessian matrix are proportional to the principle curvature of $D$. It can be shown that if $\alpha$ and $\beta$ are the larger and smaller eigenvalues respectively, and $r = \frac{\alpha}{\beta}$, we can determine whether the ratio between principal curvatures is above a set threshold by checking that

$$\frac{\text{tr}(\mathbf{H})^2}{\det(\mathbf{H})} < \frac{(r+1)^2}{r}, \tag{4.1.7}$$

where $r$ is fixed at a chosen value [22].

In order to achieve in-plane rotation invariance for the matching stage an orientation is assigned to each feature. The feature descriptor can then be calculated relative to this orientation, and thus be invariant to the rotation of the image. The scale of each feature (its position in the DoG pyramids) is used to select the closest blurred image, say $L(x, y)$, to ensure that calculations are also done in a

(a) original (b) first scale (c) second scale (d) DoG

**Figure 4.2** – Example of a DoG.

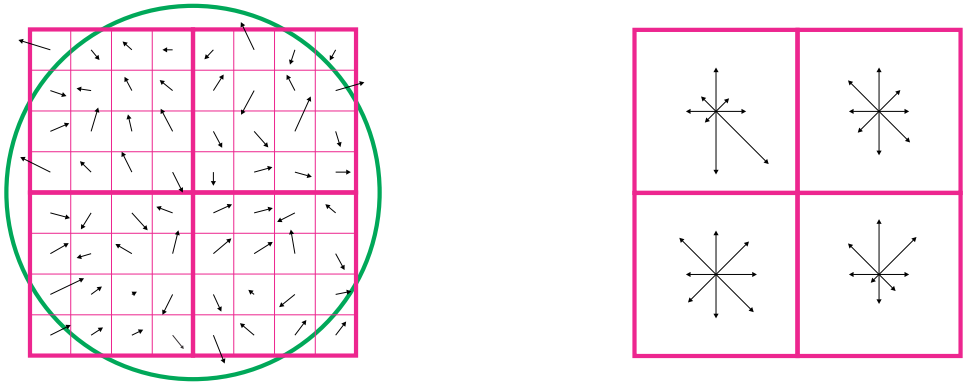
scale-invariant manner. For every pixel in $L$ we can calculate a gradient magnitude and orientation as

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + ((L(x,y+1) - L(x,y-1))^2}, \tag{4.1.8}$$

$$\theta(x,y) = \arctan\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right). \tag{4.1.9}$$

The dominant orientation of a feature is determined by the gradients of pixels around it. The orientation and magnitude of these pixels are weighted by a Gaussian, centred around the feature location, and inserted into a histogram. The Gaussian weighting is done to ensure that pixels around the edge of the region will not have a major effect on the orientation of the feature. The highest peak of the histogram is detected and refined by fitting a parabola through surrounding histogram values. The location of the peak of the parabola will be the orientation of the feature. If the height of the second highest peak is more than 80% of that of the first, a second feature is created at the same location, with an orientation corresponding to the second peak. Although this seems redundant, it contributes to the stability of matching.

Now that a location, scale and orientation are assigned to each feature, we can create a descriptor. Using the scale of the feature to select the corresponding blurred image, we once again select a region around the feature. The coordinates and orientations of the pixels in the region are rotated by the orientation of the feature to ensure invariance to image rotation. A Gaussian weighting is once again used to reduce the effect of changes in pixels around the edge of the region. Orientation histograms of subregions are then used to build the descriptor. In practice best results have been achieved by using a 4×4 array of histograms where each histogram has 8 orientation bins. Figure 4.3 depicts an example of a descriptor that uses a 2×2 array.



(a) gradient magnitudes and orientations

(b) feature descriptor

**Figure 4.3** – After gradient magnitudes have been weighted by the Gaussian (depicted as a green circle) they are grouped into histograms to create the feature descriptor.

The final step of the algorithm is to refine the descriptor in an attempt to reduce the effect of illumination. A change in contrast of an image is assumed to be similar to multiplying each pixel with a constant. By normalizing the descriptor, the effect of a linear contrast change is removed. Nonlinear illumination changes will typically cause changes in the relative magnitudes of pixels, but not their orientations. By thresholding the descriptor, the influence of large magnitudes are reduced, while emphasis remains on orientations, making the descriptor more robust to nonlinear illumination changes. After this thresholding the descriptor is normalized again.

SIFT is thorough, and for this reason very accurate and used in many applications. It is, however, very slow as it has to perform many expensive operations. The next algorithm we explore is designed to perform the same task as SIFT, at considerably faster speeds, through a number of approximations.

### 4.1.2 SURF

SURF [3] was developed to overcome the high computational complexity of SIFT. It is in many ways the same as SIFT, and the influence of SIFT is clear in the structure of the algorithm. We use the implementation included in the OpenCV 2.1 library [6] which is freely available online. Before we explain the algorithm, we provide a brief overview of integral images, or summed area tables, as they provide the basis on which SURF achieves its fast execution time.

If $I(x, y)$ is the input image, then its integral image $I_\Sigma(x, y)$ is

$$I_\Sigma(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j). \tag{4.1.10}$$

With the integral image the sum of the pixels in any rectangular region of the original image can be calculated by using only four additions. Figure 4.4 clarifies this concept.

When the integral image is used, certain convolutions can be performed very fast and, seeing that box-type image filters are essentially convolutions, we can speed up the filtering by choosing the filter mask carefully.

The SURF algorithm begins by considering scale space pyramids, similar to SIFT, consisting of

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{4.1.11}$$

with $I(x, y)$ the original image and $G(x, y, \sigma)$ given in Equation 4.1.2. Instead of using DoGs, features are selected by searching for extrema in the determinant of a Hessian matrix. The determinant of the Hessian is also used to determine the scale of the feature. Here the Hessian is defined as

$$\mathbf{H} = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{yx}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix}, \tag{4.1.12}$$

where the entries are second order derivatives of the Gaussian-blurred image $L(x, y, \sigma)$. Instead of first blurring the image and then calculating the Hessians, the filter masks depicted in the top row of Figure 4.5 can be applied in a single step. To further improve the speed of the convolution, SURF approximates these filters, as shown in the bottom row of Figure 4.5, enabling the use of integral

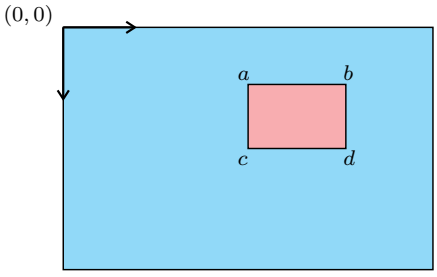



**Figure 4.4** – The integral image is used to calculate the sum of intensities in any rectangular region efficiently as $s = I_\Sigma(a) - I_\Sigma(b) - I_\Sigma(c) + I_\Sigma(d)$.

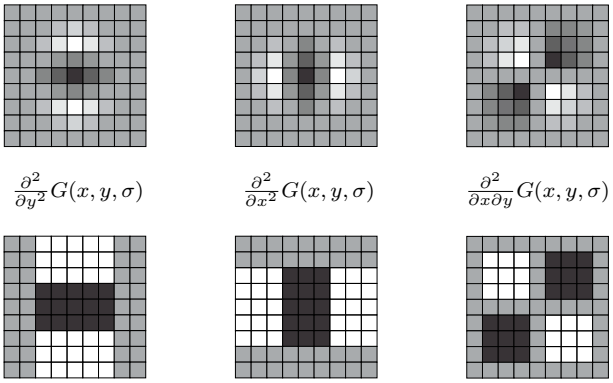



**Figure 4.5** – Normal filter masks (top) with their approximations (bottom). In this representation dark elements will be negative, light positive and grey zero.

images. In doing so the computational cost becomes independent of filter size. It has been shown [3] that this approximation does not have a large adverse effect on accuracy.

We denote the approximated Hessian as

$$\mathbf{H}_{\text{approx}} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}. \tag{4.1.13}$$

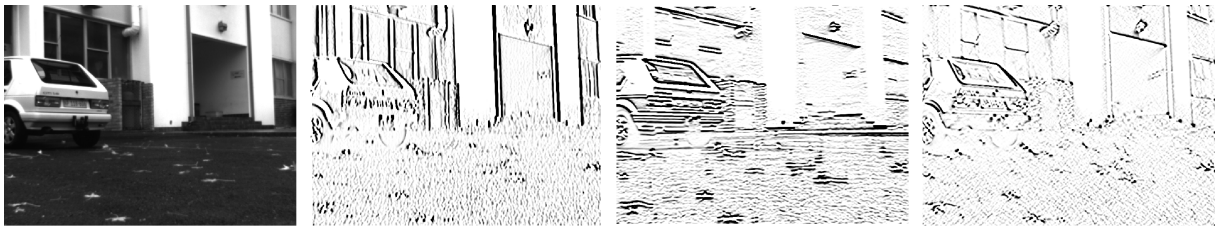
The determinant is

$$\det(\mathbf{H}_{\text{approx}}) = D_{xx}D_{yy} - (wD_{xy})^2, \tag{4.1.14}$$

where $w \approx 0.9$ is a weight which conserves the energy between the Gaussian kernel and the approximated Gaussian kernel [3]. An example of the Hessian responses on a typical image are shown in Figure 4.6.

Since the size of the filter does not influence execution time, the filter size is increased instead of down-sampling the image when the scale space pyramid is created. Scale changes brought on by increasing the size of the filter mask may be crude, so to achieve finer levels of scale change the image is up-sampled by a factor of two. Figure 4.7 depicts the upscaling of two of the filter masks. We once again divide different scales into octaves so that every octave represents a scale change of two.

Feature localization up to sub-pixel accuracy in scale space is done, as for SIFT, by using Equation 4.1.5.



(a) original (b) $\frac{\partial^2}{\partial x^2}G(x,y,\sigma)$ (c) $\frac{\partial^2}{\partial y^2}G(x,y,\sigma)$ (d) $\frac{\partial^2}{\partial x \partial y}G(x,y,\sigma)$

**Figure 4.6** – Examples of Hessian responses. Note how the second order derivative in the $x$ direction has a strong response over vertical lines and in the $y$ direction a strong response over horizontal lines. When the derivative is taken in both directions diagonal lines are highlighted.

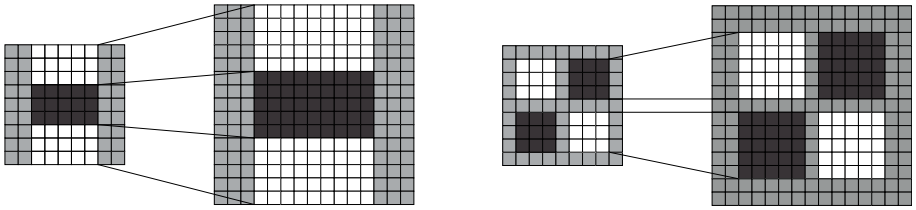


**Figure 4.7** – Upscaling the filter from 9×9 to 15×15. To ensure the presence of a central pixel the dark lobe (left) should only be increased by an even number of rows and columns.

As in SIFT, an orientation is assigned to every feature to enable invariance to image rotation. Instead of looking at image gradients, we can use the integral image to calculate Haar wavelet responses in the horizontal and vertical directions. The size of the wavelet is dependent on the scale of the feature and is set to $4s \times 4s$ where $s$ is the scale of the feature. Haar wavelet responses are calculated for a circular region, with radius $6s$, around each feature. The responses are then weighted with a Gaussian around the feature.

With the wavelet responses, a dominant orientation is assigned using a sliding orientation window. The horizontal and vertical responses within the window are summed, and the largest is the dominant orientation of the feature. Figure 4.8 depicts such a window with a dominant orientation.

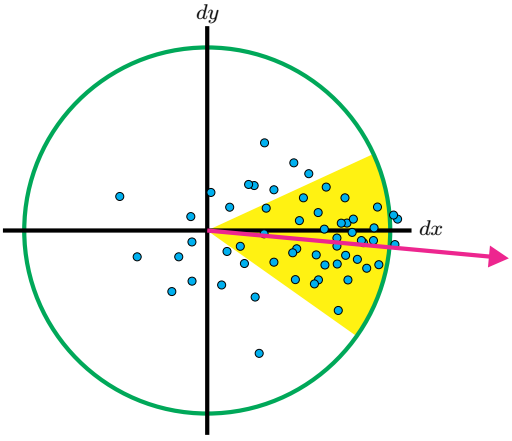



**Figure 4.8** – By sliding the window (yellow) around, the dominant orientation is determined for a SURF feature by summing Haar wavelet responses.

To construct a descriptor, a square region around the feature is selected. The orientation of the region is normalized according to the orientation of the feature and is set to have a size relative to the scale, normally $20s$. The region is split into 4×4 subregions and Haar wavelet responses are calculated relative to the orientation of the feature. We thus obtain a horizontal response $dx$ and a vertical response $dy$ for each pixel in the region. As before, these responses are weighted by a Gaussian. For every subregion we then calculate a four-dimensional vector containing sums of the responses,

$$\mathbf{d} = \left[\sum dx \quad \sum |dx| \quad \sum dy \quad \sum |dy|\right]^T. \tag{4.1.15}$$

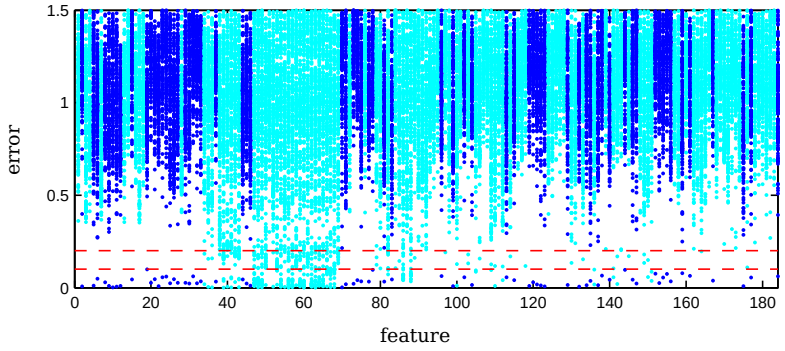
By summing the absolute values of the responses we include information about the polarity of intensity changes. Finally the descriptor is normalized in order to achieve invariance to contrast. By combining 16 subregion descriptors, we arrive at a 64-dimensional vector.

### 4.1.3 Matching

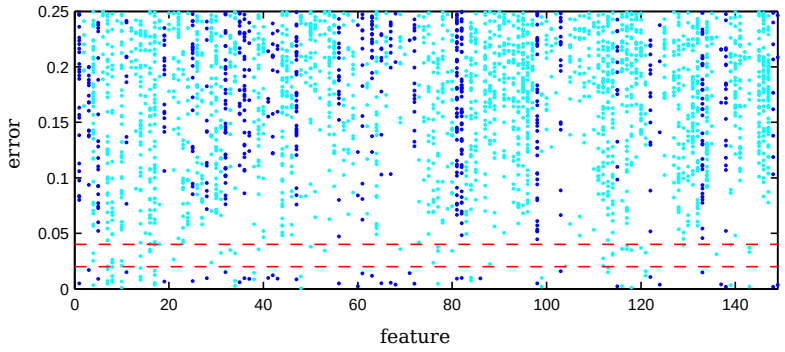
Once features have been identified in images, using either SIFT or SURF, we use the descriptors to match features from different images to one another, firstly between left and right images of the current time step and then features at the current time step with those observed previously. Descriptors for both SIFT and SURF are designed in such a way that a simple nearest neighbour search on the Euclidean error between descriptors can be used to find matches. Care has to be taken when this is done, as many incorrect matches can be made because of ambiguous features. We take the following simple steps to eliminate what can be called obvious mismatches.

- **Epipolar constraint:** When we match between left and right images from the same time step, we can rectify the images and ensure that feature matches have the same vertical coordinates (see Section 3.2.3).
- **3D location relative to the sensor:** Feature matches from the same time step can be triangulated to 3D (Section 3.2.4). When we have the 3D locations, we can eliminate those that fall outside the field of view. If a feature is, for instance, behind the cameras there must be a mistake in the matching and the feature is discarded. We also enforce a maximum and minimum distance threshold.
- **Bijectivity:** If a feature in one image is the best match for more than one feature in the other image we discard the match, rather than make a potentially incorrect decision.
- **Consistency check:** When matching features between two images we compare each descriptor from the first image with all the descriptors of the second image to find the best match. By repeating the process the other way round, using the second image as a reference, we can obtain a second set of matches. By discarding inconsistent matches we reject some ambiguous features. This process can be very effective, but roughly doubles the execution time of the matching.
- **Descriptor double error threshold:** We extend matching with the descriptors slightly by adding the constraint that the second best match to a feature must be significantly worse than the best match. This eliminates ambiguous features, although it may also result in the loss of many correct matches. The two thresholds, one for the best match and one for the second best match, must therefore be selected carefully. Figure 4.9 depicts an example of double error threshold matching.
- **Left-left right-right consistency check:** Once we have matched the left and right image features, we can match features in the current left image with features from previous left images, and features in the current right image to previous right image features. Only features that match consistently for left images and right images are kept.

Depending on the required application, we sometimes use all of the above-mentioned methods to get the best possible set of matches. However, if execution time is more important, we skip some of the steps (especially the consistency checks). Figure 4.10 depicts the result of our matching process on typical images from two time steps. We find that the majority of the feature matches are correct,




(a) SIFT feature matching errors




(b) SURF feature matching errors

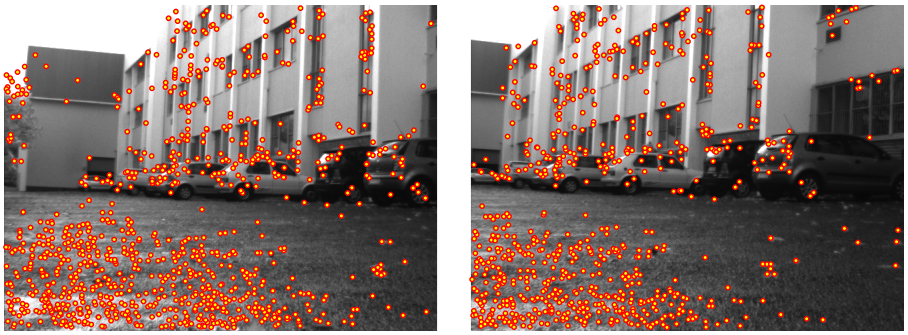
**Figure 4.9** – Euclidean errors between descriptors from two typical images, with two thresholds. If the smallest error is below the lower threshold and the second smallest error above the higher threshold a match is assigned (blue), otherwise no match is assigned for that feature (cyan).

but that a few incorrect matches can slip through. We return to this in Section 4.3, but first we compare SIFT and SURF to establish which one would be best suited for our purpose.

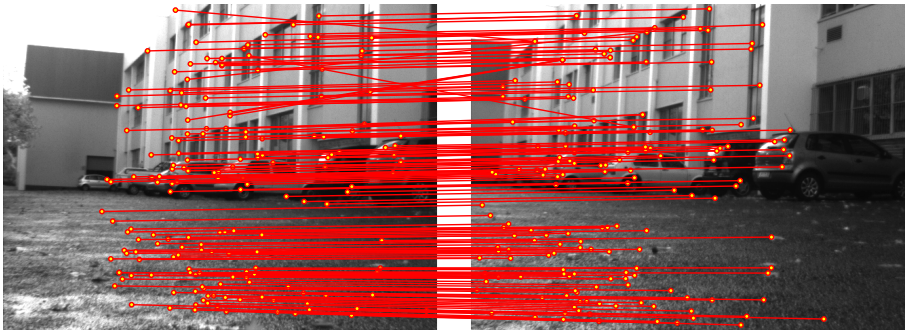
### 4.1.4 Comparison

We conclude this section with a brief comparison between SIFT and SURF in terms of execution times and accuracy. Execution times depend on many factors other than algorithm complexity and are therefore difficult to compare reliably, and this should be kept in mind for the times we provide here. The detection and matching of features in left and right images at one time step are grouped into one step, which we call the measurement step because this step provides a 3D measurement of landmarks. These landmarks can then be matched to landmarks measured at previous time steps, giving us putative matches. For comparison, we evaluated the putative matches by hand and counted the ones considered safe to use. The results from a dataset of 300 images are summarized in Table 4.1.

These results confirm our previous statements that SIFT is typically more accurate while SURF is faster. For SIFT the measurements are smaller, but we can use a larger percentage of them, which means that fewer are discarded. For SIFT less than 1 in a 100 feature matches were incorrect, in comparison to SURF where there were more than 8 in 100. The execution time benefit of SURF does, however, make up for the loss in accuracy. From these results the conclusion is clear: if we require real-time implementation, SURF is better, but if we can post-process a dataset and execution



(a) SIFT features



(b) SIFT matches

**Figure 4.10** – SIFT feature matches between two images taken at different time steps. Note that despite all our efforts incorrect matches still occur.

| Average over dataset | SIFT | SURF |
|---|---|---|
| Execution time of measurement (ms) | 3083 | 316 |
| Number of landmarks in measurement | 119.6 | 178.3 |
| Number of putative matches | 56.2 | 47.1 |
| Number of correct putative matches | 55.7 | 43.1 |

**Table 4.1** – Comparison between SIFT and SURF. All the values are given as an average over 300 time steps. The execution time of a measurement includes feature detection (in both left and right images) and matching (between left and right image features). The number of putative matches are the size of the initial set of landmark matches between two consecutive measurements obtained by using the matching methods discussed in the previous section. From the putative set we count the correct matches to evaluate the accuracy of the matching.

time is not a consideration, SIFT will give accurate results.

## 4.2 Measurement uncertainty

Once we have measured feature matches we have to characterize the measurement noise in order to find landmark mismatches (as we see in Section 4.3). A noise model is also necessary to enable optimal estimation of their locations (which is the topic of the next chapter). We first model the noise in image coordinates, and then discuss the method we use to transform that noise to robot coordinates.

We assume that a feature will be measured with zero mean Gaussian noise on the image plane, that is

$$\mathbf{x}_{\text{im}} = \begin{bmatrix} x_L \\ y_L \\ x_R \\ y_R \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{N}), \tag{4.2.1}$$

where $x_L$, $y_L$, $x_R$ and $y_R$ are the image coordinates of the feature in the left and right image respectively, and

$$\mathbf{N} = \begin{bmatrix} \sigma_{x_L}^2 & 0 & 0 & 0 \\ 0 & \sigma_{y_L}^2 & 0 & 0 \\ 0 & 0 & \sigma_{x_R}^2 & 0 \\ 0 & 0 & 0 & \sigma_{y_R}^2 \end{bmatrix}. \tag{4.2.2}$$

By $\mathcal{N}(\mathbf{0}, \mathbf{N})$ we mean a sample drawn from the normal distribution with mean $\mathbf{0}$ and covariance matrix $\mathbf{N}$. For the sake of simplicity we assume that the noise in the different image coordinates is statistically independent. The $\sigma$ parameters are camera and lens specific and have to be calibrated through practical tests. From the previous chapter we have the transformation from image coordinates in pixels to robot coordinates in metres (Equation 3.2.14). We use a Gaussian noise model with zero mean and covariance matrix $\mathbf{Q}$ with this transformation, that is

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} \frac{fb}{x_L - x_R} \\ \frac{(x_L - p_x)b}{x_L - x_R} - \frac{b}{2} \\ \frac{(0.5(y_L + y_R) - p_y)b}{x_L - x_R} \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{Q}). \tag{4.2.3}$$

Since the transformation given in Equation 3.2.14 is nonlinear, we approximate the noise transformation with a first order Taylor expansion (a trick we will use numerous times throughout this thesis). This approximation is described by the Jacobian of the transformation evaluated at the measurement, i.e.

$$\mathbf{W} = \begin{bmatrix} \frac{\partial x_r}{\partial x_L} & \frac{\partial x_r}{\partial y_L} & \frac{\partial x_r}{\partial x_R} & \frac{\partial x_r}{\partial y_R} \\ \frac{\partial y_r}{\partial x_L} & \frac{\partial y_r}{\partial y_L} & \frac{\partial y_r}{\partial x_R} & \frac{\partial y_r}{\partial y_R} \\ \frac{\partial z_r}{\partial x_L} & \frac{\partial z_r}{\partial y_L} & \frac{\partial z_r}{\partial x_R} & \frac{\partial z_r}{\partial y_R} \end{bmatrix} = \begin{bmatrix} \frac{-bf}{(x_L - x_R)^2} & \frac{bf}{(x_L - x_R)^2} & 0 & 0 \\ \frac{b(p_x - x_R)}{(x_L - x_R)^2} & \frac{-b(px - x_L)}{(x_L - x_R)^2} & 0 & 0 \\ \frac{-b(y_L - 2p_y + y_R)}{2(x_L - x_R)^2} & \frac{b(y_L - 2p_y + y_R)}{2(x_L - x_R)^2} & \frac{b}{2(x_L - x_R)} & \frac{b}{2(x_L - x_R)} \end{bmatrix}. \tag{4.2.4}$$

With this Jacobian we use the linear transformation of multivariate Gaussians [29, p. 159], that is

$$\mathbf{Q} = \mathbf{W}\mathbf{N}\mathbf{W}^T, \tag{4.2.5}$$

to transform the noise characterization from image coordinates to robot coordinates.

In order to see the effect of this linearization, and evaluate the accuracy of our assumptions, we measured the noise on typical landmarks. Figure 4.11 depicts, for each of two of the several landmarks tested, 500 measurements in robot coordinates with a confidence ellipse obtained from a Gaussian fit of the measurements. We compared the measured noise in the landmarks with the noise model in Equation 4.2.5, and found them to be consistently similar. These tests enabled us to calibrate the values of $\sigma_{x_L}$, $\sigma_{y_L}$, $\sigma_{x_R}$ and $\sigma_{y_R}$ and confirm that our approximations are reasonable.

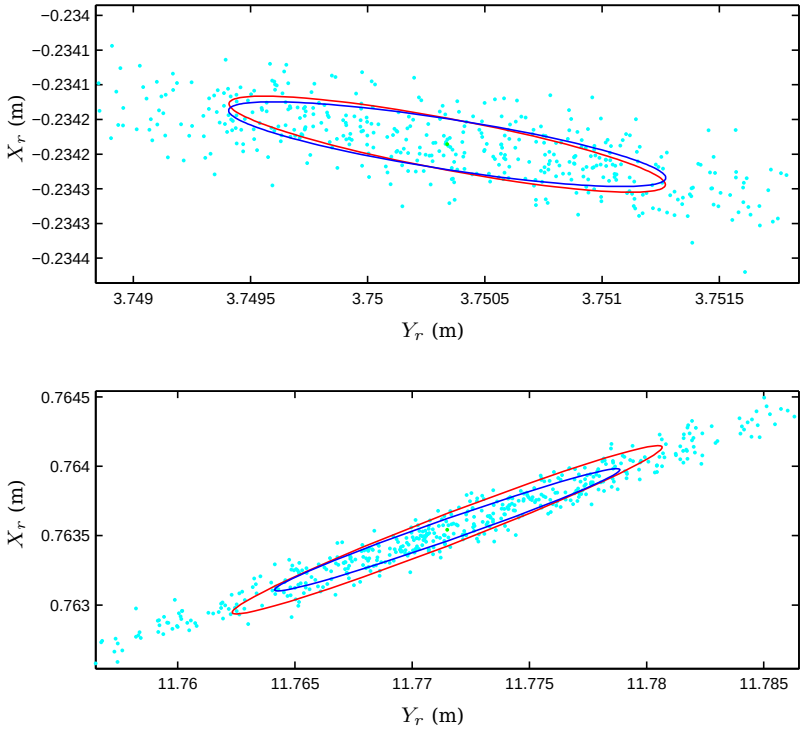



**Figure 4.11** – A comparison between noise of measured landmarks (cyan dots with fitted confidence ellipses in blue) and calculated covariance (red confidence ellipses).

## 4.3 Outlier detection

Matching feature descriptors can be sufficient for finding putative landmark correspondences. However, there will usually be some mismatches that can adversely affect the accuracy of a SLAM system. RANSAC is a popular technique for identifying and removing such errors. In this section we give an overview of the algorithm and explain how it is commonly used with the fundamental matrix. We then propose a new way of using the RANSAC framework with the measurement model derived in the previous section, and argue that it is more suited for SLAM.

### 4.3.1 The RANSAC algorithm

Random sample consensus (RANSAC) [13] is a technique for estimating model parameters from data potentially containing outliers. Its basic structure can be expressed as two steps repeated $k$ times:

1. draw a random minimal sample from the data and fit model parameters to the sample;
2. find a consensus set from the data that corresponds to the proposed model within a fixed threshold or, in other words, find the data points that could have been generated from the fitted model.

Upon completion, inliers are identified as the largest consensus set found. The model parameters can then be re-estimated by using the entire set of inliers.

The number of iterations is commonly calculated as

$$k = \frac{\log(1-p)}{\log(1-w^n)}, \tag{4.3.1}$$

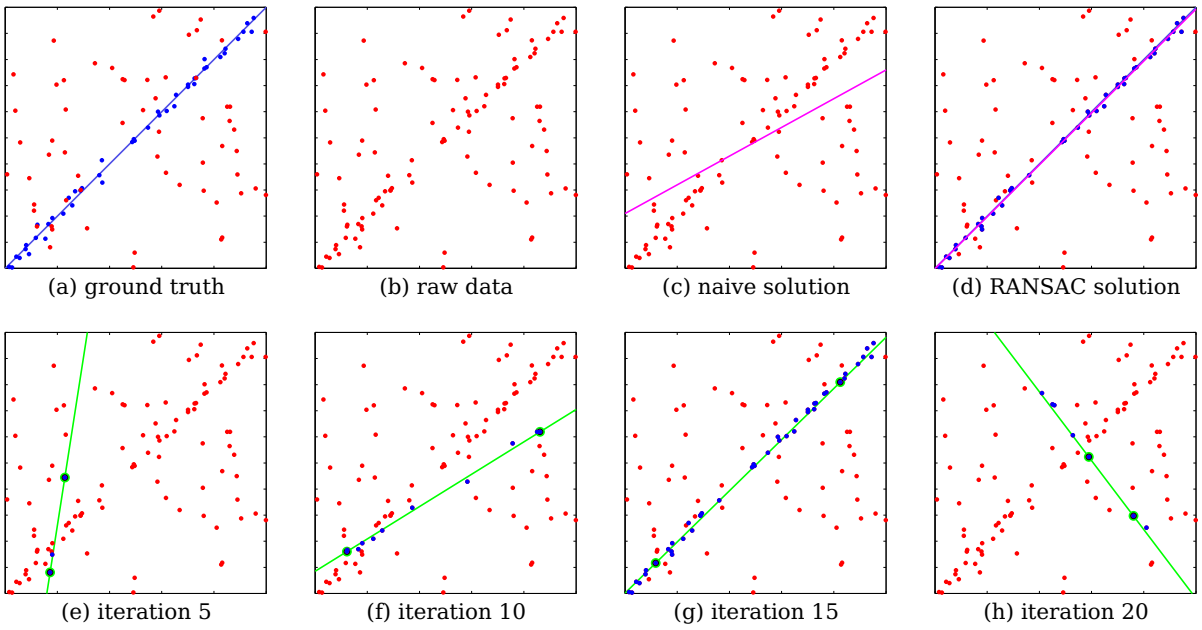



**Figure 4.12** – An example of the use of RANSAC. Data points are sampled from a straight line with Gaussian noise added (blue points in (a)) and outliers are randomly added (red points in (a)) giving us the dataset in (b). A naive solution can be found if we solve using least squares. However, as shown in (c), this solution is corrupted by the outliers. By first identifying inliers with RANSAC a more accurate solution can be calculated with least squares, as shown in (d). Several RANSAC iterations are shown in (e-f) with identified inliers (blue), outliers (red) and the proposed model (green).

with $p$ the desired probability that the algorithm will produce a good result, $w$ the probability of choosing an inlier when a single point is drawn from the data and $n$ the size of the sample.

The RANSAC algorithm is illustrated in Figure 4.12 by means of a simple example. Data is generated by using a straight line with added noise and outliers are randomly added. In this case, the model is the equation for a line, i.e. $y = mx + c$, and the consensus measure is distance to the line. RANSAC accurately identifies inliers and produces a result superior to a naive least squares fit.

When outliers need to be identified from image features correspondences, a standard approach is to use the fundamental matrix as a model for RANSAC [41]. The fundamental matrix is a $3 \times 3$ matrix that encapsulates the epipolar geometry between two camera views, as discussed in the previous chapter. The Sampson distance [17] can be used to determine consensus of data points to the proposed model. This approach can be highly effective in some cases, but it is not ideal for SLAM applications. Its first limitation is the use of image coordinates and not 3D world coordinates as those found in a typical SLAM map. Secondly, at least seven correspondences are needed in order to calculate a fundamental matrix and, from Equation 4.3.1, this can mean that a large number of iterations are required. The third limitation is the possibility that mismatches can be classified as inliers. If mismatched features are by chance close to corresponding epipolar lines, they will agree with the model and will not be identified as outliers.

In order to overcome these limitations we propose a different model, based on the 3D locations of the landmarks, and consensus measure, based on the measurement noise model, to be used in a RANSAC framework.

### 4.3.2 Improved consensus measure

In an effort to overcome the limitations of using the fundamental matrix with RANSAC, we prefer to work with the 3D coordinates of the measurement. For every new measurement, there exists a 3D translation vector and rotation matrix that relate the measured landmarks to corresponding landmarks in the map already built from previous measurements. This rigid transformation can be calculated easily, using for example the least squares method of Umeyama [37], and used as a RANSAC model. Only three 3D point correspondences are needed and this reduced sample size results in a smaller value for $k$ (defined in Equation 4.3.1) and therefore a faster execution time.

A simple way to establish consensus of a 3D point correspondence to the proposed model would be to calculate the Euclidean distance between the two points after one has been translated and rotated with the model parameters. This approach, however, produces sub-optimal results because it does not consider the nature of the measurement noise. We found the likelihood that two measurements are of the same point, explained below, to be a far better measure of consensus.

Suppose at time $t$ our measurement of a point is $\boldsymbol{\mu}_1$, with an associated covariance matrix $\boldsymbol{\Sigma}_1$ describing the uncertainty. At time $t+1$ we take a second measurement $\boldsymbol{\mu}_2$ with covariance matrix $\boldsymbol{\Sigma}_2$. Let us assume that one of these measurements (along with its uncertainty) is rotated and translated by the RANSAC model. If we now postulate that these two measurements are of the same landmark, we wish to calculate the joint probability

$$p_c = p(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2). \tag{4.3.2}$$

If $\mathbf{r}$ denotes the (unknown) true location of the landmark, and we assume the prior $p(\mathbf{r})$ to be uniform over $\mathbf{V} = \mathbb{R}^3$, then

$$\begin{aligned} p_c &= \int_{\mathbf{V}} p(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mathbf{r})\, d\mathbf{r} && (4.3.3)\\ &= \int_{\mathbf{V}} p(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \,|\, \mathbf{r})\, p(\mathbf{r})\, d\mathbf{r} && (4.3.4)\\ &\propto \int_{\mathbf{V}} p(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \,|\, \mathbf{r})\, d\mathbf{r}. && (4.3.5) \end{aligned}$$

We assume the two measurements to be conditionally independent since they are made at different time steps. Also, since we use a Gaussian noise model for each measurement, we have

$$\begin{aligned} p_c &\propto \int_{\mathbf{V}} p(\boldsymbol{\mu}_1 \,|\, \mathbf{r})\, p(\boldsymbol{\mu}_2 \,|\, \mathbf{r})\, d\mathbf{r} && (4.3.6)\\ &= \int_{\mathbf{V}} \frac{1}{(2\pi)^{\frac{3}{2}} |\boldsymbol{\Sigma}_1|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{\mu}_1 - \mathbf{r})^T \boldsymbol{\Sigma}_1^{-1} (\boldsymbol{\mu}_1 - \mathbf{r})} \frac{1}{(2\pi)^{\frac{3}{2}} |\boldsymbol{\Sigma}_2|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{\mu}_2 - \mathbf{r})^T \boldsymbol{\Sigma}_2^{-1} (\boldsymbol{\mu}_2 - \mathbf{r})}\, d\mathbf{r} && (4.3.7)\\ &= \frac{1}{(2\pi)^{\frac{3}{2}}} \frac{|\boldsymbol{\Sigma}|^{\frac{1}{2}}}{|\boldsymbol{\Sigma}_1|^{\frac{1}{2}} |\boldsymbol{\Sigma}_2|^{\frac{1}{2}}} e^{-\frac{1}{2}s} \int_{\mathbf{V}} \frac{1}{(2\pi)^{\frac{3}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{r} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{r} - \boldsymbol{\mu})}\, d\mathbf{r}, && (4.3.8) \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{\Sigma}^{-1} &= \boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}, && (4.3.9)\\ \boldsymbol{\mu} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2), && (4.3.10) \end{aligned}$$

and

$$s = \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}_1^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^T \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2 - (\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\mu}_2^T \boldsymbol{\Sigma}_2^{-1})\, \boldsymbol{\Sigma}^{-1} (\boldsymbol{\Sigma}_1^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2). \tag{4.3.11}$$

The integral part of Equation 4.3.8 evaluates to 1, yielding

$$p_c \propto \frac{|\mathbf{\Sigma}_1^{-1}|^{\frac{1}{2}}|\mathbf{\Sigma}_2^{-1}|^{\frac{1}{2}}}{|\mathbf{\Sigma}_1^{-1}+\mathbf{\Sigma}_2^{-1}|^{\frac{1}{2}}}\, e^{-\frac{1}{2}s}, \tag{4.3.12}$$

with $s$ stated in Equation 4.3.11. The expression on the right hand side of Equation 4.3.12 can now be used as a consensus measure with RANSAC.

An example of the benefit of using this consensus measure is illustrated in Figure 4.13 by means of two measurements of a point. By using the result above, the pair on the right would be viewed as being more likely to be two measurements of the same point, despite having a larger Euclidean difference in their means, than the pair on the left. In the next section we explain how we go about using this consensus measure with RANSAC to detect incorrect feature matches.

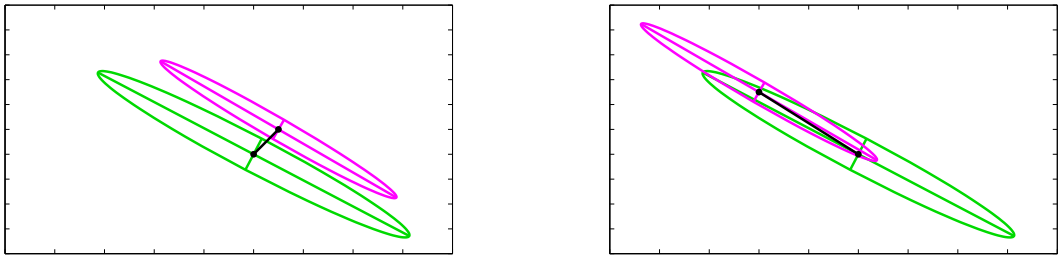


**Figure 4.13** – Two pairs of measurements depicted as Gaussian confidence ellipses. Although the means of the pair on the left are closer to each other, the pair on the right have a higher likelihood of being measurements of the same point.

### 4.3.3 Outlier detection with the improved consensus measure

With our proposed consensus measure and the RANSAC framework, we can formulate an outlier detection scheme. The algorithm operates under the assumption that the structure of two sets of 3D points remains unchanged, even when measured at different time steps and from different view points.

The inputs of the algorithm are any two sets of 3D points of the form

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \qquad \text{and} \qquad \mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m], \tag{4.3.13}$$

and corresponding covariance matrices

$$\mathbf{C} = [\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_n] \qquad \text{and} \qquad \mathbf{P} = [\mathbf{P}_1, \mathbf{P}_2, \cdots, \mathbf{P}_m], \tag{4.3.14}$$

respectively. The putative correspondences that the algorithm tests are contained in the correspondence vector $\mathbf{c}$ such that if $c_i = j$, $\mathbf{x}_i$ corresponds to $\mathbf{y}_j$. An entry of zero in $\mathbf{c}$ indicates that there is no match for that specific point in $\mathbf{x}$.

Our scheme is discussed next, with reference to Algorithm 4.1 below.

- **Line 2:** The algorithm is executed $k$ times, with $k$ calculated according to Equation 4.3.1.
- **Lines 3 and 4:** According to the basic RANSAC structure, we draw a random minimal sample from the data. In our case the sample consists of three entries. The correspondence vector $\mathbf{c}$ is used to find the entries in $\mathbf{y}$ that are matched to the sample from $\mathbf{x}$.

**Algorithm 4.1** probabilistic_outlier_removal($\mathbf{x}$, $\mathbf{C}$, $\mathbf{y}$, $\mathbf{P}$, $\mathbf{c}$)

1: best_inlier_count := $0$
2: **for** $k$ iterations **do**
3: draw random sample $\mathbf{x}_s$ of three points, that has nonzero entries in $\mathbf{c}$, from $\mathbf{x}$
4: draw sample $\mathbf{y}_s$ from $\mathbf{y}$ corresponding to $\mathbf{x}_s$, according to $\mathbf{c}$
5: $[\mathbf{R}, \mathbf{t}, s]$ := umeyama$(\mathbf{x}_s, \mathbf{y}_s)$
6: inlier_count := $0$
7: $\mathbf{c}_{\text{temp}} := \mathbf{c}$
8: **if** $s \approx 1$ **then**
9: **for** $i \in \{1, 2, \ldots, n\}$ **do**
10: $j := c_{\text{temp},i}$
11: **if** $j > 0$ **then**
12: $\mathbf{x}_{\text{temp}} := \mathbf{R}\mathbf{x}_i + \mathbf{t}$
13: $\mathbf{C}_{\text{temp}} := \mathbf{R}\mathbf{C}_i\mathbf{R}^T$
14: $p := f(\mathbf{x}_{\text{temp}}, \mathbf{C}_{\text{temp}}, \mathbf{y}_j, \mathbf{P}_j)$
15: **if** $p > p_{\text{threshold}}$ **then**
16: inlier_count := inlier_count $+ 1$
17: **else**
18: $\mathbf{c}_{\text{temp},i} = 0$
19: **end if**
20: **end if**
21: **end for**
22: **if** inlier_count $>$ best_inlier_count **then**
23: $\mathbf{c}_{\text{best}} := \mathbf{c}_{\text{temp}}$
24: best_inlier_count := inlier_count
25: **end if**
26: **end if**
27: **end for**
28: **return** $\mathbf{c}_{\text{best}}$

- **Line 5:** We use the method described by Umeyama [37] to find a similarity transformation between $\mathbf{x}_s$ and $\mathbf{y}_s$. This method provides us with a scale factor $s$, a 3D rotation matrix $\mathbf{R}$ and a 3D translation vector $\mathbf{t}$ so that

$$\mathbf{y}_s = s\mathbf{R}\mathbf{x}_s + \mathbf{t}, \tag{4.3.15}$$

  in a least squares sense. We use this transformation as our RANSAC model.

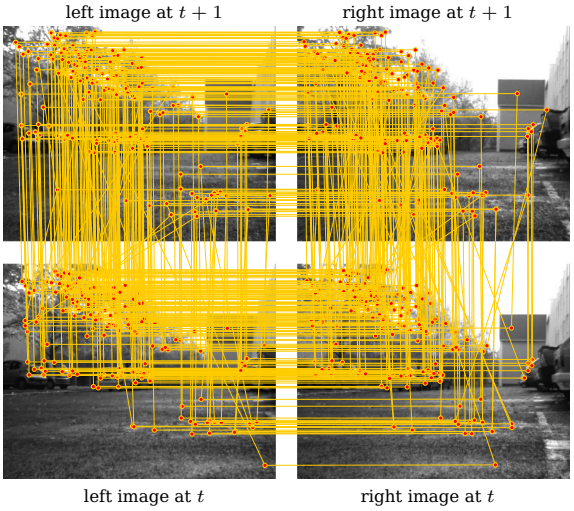
- **Lines 6 and 7:** In order to test the proposed model against the data, we count the number of data points that correspond with it. We define a temporary correspondence vector for every proposed model in order to store the inliers the algorithm identifies with that model.

- **Lines 8 to 10:** Since the scale of the world cannot change we can reject a model immediately if the scale is not close to 1. If it is, we enter a loop over the data with the index $i$ corresponding to the entries of $\mathbf{x}$ and index $j$ to the corresponding entries of $\mathbf{y}$.

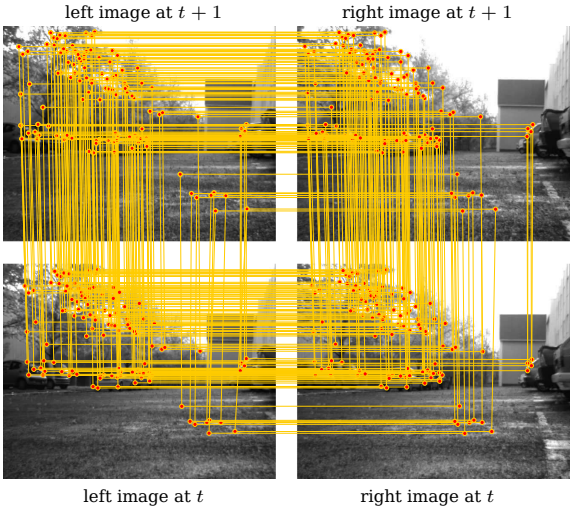
- **Lines 11 to 14:** Using the candidate model, we transform each putatively matched entry of $\mathbf{x}$ along with its covariance matrix. Ideally $\mathbf{x}_{\text{temp}}$ will be equal to the corresponding $\mathbf{y}_j$. Our proposed consensus measure can now be used to calculate how well the point in question fits the proposed model. The function $f$ refers to the right hand side of Equation 4.3.12.
- **Lines 15 to 19:** If the value of $p$ is greater than some threshold, the point is counted as an inlier. If it is not the case, we update the temporary correspondence vector to mark that the point does not have a match.
- **Lines 22 to 25:** Once all the entries of $\mathbf{x}$ have been tested we check whether the proposed model has a larger number of inliers than the previous best. If it does, the best inlier score is updated and the temporary correspondence vector is stored.
- **Line 28:** Upon completion the algorithm returns the correspondence vector corresponding to the model with the most inliers.

Figure 4.14 depicts the benefit of using our improved outlier removal scheme. We set the matching thresholds to allow more mistakes than usual to demonstrate the method better (the set of putative matches contains more outliers than it typically would). We found that the standard fundamental matrix RANSAC method for detecting outliers can be highly sensitive to the threshold used when building a consensus set. It can easily be too permissive, so that the method incorrectly classifies mismatches as inliers, or too strict, resulting in a very small number of correct matches. Our method, on the other hand, is much less sensitive and we find it to be very effective at retaining almost all correct matches without erroneously allowing mismatches.

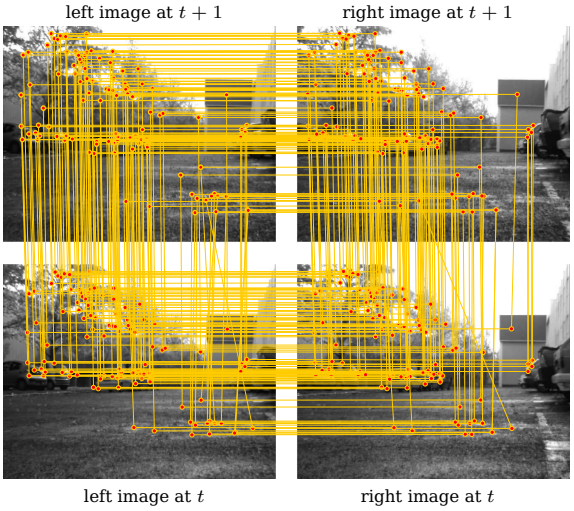
Accurate landmark identification is vital for any SLAM system but, before we can use these landmarks with SLAM, we first need to investigate possible solutions to the SLAM problem and we do so in the next chapter.



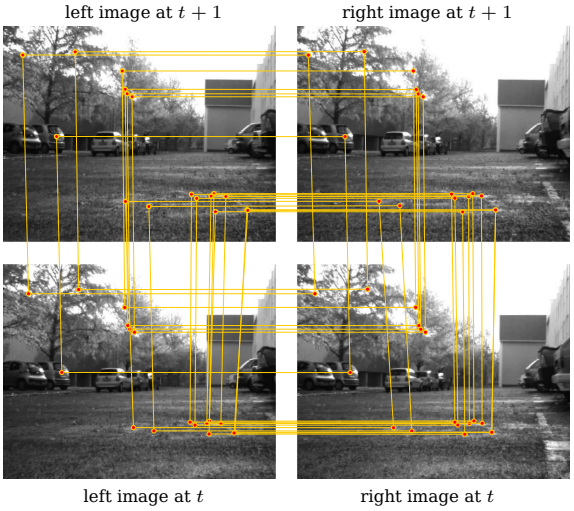

(a) putative matches




(b) inlier matches with our outlier removal scheme




(c) inlier matches with fundamental matrix RANSAC, with a permissive threshold




(d) inlier matches with fundamental matrix RANSAC, with a strict threshold

**Figure 4.14** – A comparison between the two RANSAC methods we considered on a set of matches containing many errors (a). The method we propose (b) provides a larger set of inliers, while still removing outliers effectively, when compared to the standard approach (c,d).

# Chapter 5

# SLAM algorithms

This chapter moves the discussion towards the SLAM problem in an attempt to answer the question: if we can track image landmarks over time, how do we estimate the location of the robot and build a map containing these landmarks?

We begin by formulating the SLAM problem and the relationship between the input variables and the states we want to estimate. We choose three algorithms to solve this problem, namely EKF SLAM, FastSLAM and FastSLAM 2. EKF SLAM linearizes the system in order to utilize the Kalman filter, while FastSLAM and FastSLAM 2 are based on the Rao-Blackwellized particle filter. This chapter provides a detailed explanation of each of these algorithms.

## 5.1 The SLAM problem

Simultaneous localization and mapping (SLAM) is a technique used by a mobile robot to build a map of an unknown environment while simultaneously tracking its own motion in this partially built map. At every time step it estimates the posterior

$$p(\mathbf{x}_t, \mathbf{m} \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}), \tag{5.1.1}$$

with $\mathbf{x}_t$ the pose of the robot, $\mathbf{m}$ a map of the environment, and $\mathbf{z}_{1:t}$ and $\mathbf{u}_{1:t}$ respectively the measurements made by the robot and the control inputs given to the robot from the first time step up to the current time step.

In practice it is very difficult to solve the SLAM problem, mainly because of its high dimensionality (and the discrete nature of the landmark correspondence decisions, but that has been discussed in the previous chapter). Many algorithms and methods have been proposed in an effort to solve SLAM [34]. We investigate three of these algorithms: EKF SLAM (based on the Kalman filter), FastSLAM and FastSLAM 2 (both based on the particle filter). EKF SLAM is historically important as it was the first major break-through in the search for a practical solution to SLAM [12]. Although it has several shortcomings it is still, in our view, relevant to the field. FastSLAM and FastSLAM 2 are more recent and have shown very promising results in practical state-of-the-art systems [15].

Before we look at SLAM, we first need to establish the mathematical relationship between the variables that we want to estimate and the variables that we know. Most of the equations in this chapter are given for 3D or 6 degree-of-freedom SLAM. The simplification to 2D is almost always a case of simply omitting the rows and columns of the vectors and matrices relating to redundant states.

## 5.2 Vehicle and measurement model

It is important to have a motion model of the robot and describe the relationship between it and the measurements. This model can be as simple as a constant velocity model that uses no control input information (as is the case with some vision-based SLAM systems [28]). We assume we can obtain some useful information from the control input to the robot, and derive a motion model that encapsulates this information. We also provide a measurement model that connects the stereo geometry from Chapter 3 with our SLAM systems.

We define a new axis system called the world axes and denote them with a subscript $w$. This axis system is static relative to the environment, while the previously defined robot axes move with the robot, as depicted in Figure 5.1. The robot's state vector is defined as the location and orientation of the robot in the world axis system, and is given by

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \theta_t \\ \phi_t \\ \psi_t \end{bmatrix}, \tag{5.2.1}$$

with $x_t$, $y_t$ and $z_t$ the location of the robot at time $t$. The pose of simple terrestrial robots can be described adequately with these six states. It is, however, important to note that when one works with more complex systems, additional states may be required for accurate estimation. Although these extra states change the motion model, it does not change the approach we follow conceptually.

We define the orientation of the robot by means of Euler angles. For a given orientation, the robot axes are first rotated about the $Z_w$ axis by $\psi_t$ radians, then about the new $Y$ axis by $\phi_t$ radians and finally about the new $X$ axis by $\theta_t$ radians. This rotation is described mathematically by the rotation matrix

$$\mathbf{R}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_t) & -\sin(\theta_t) \\ 0 & \sin(\theta_t) & \cos(\theta_t) \end{bmatrix} \begin{bmatrix} \cos(\phi_t) & 0 & \sin(\phi_t) \\ 0 & 1 & 0 \\ -\sin(\phi_t) & 0 & \cos(\phi_t) \end{bmatrix} \begin{bmatrix} \cos(\psi_t) & -\sin(\psi_t) & 0 \\ \sin(\psi_t) & \cos(\psi_t) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.2.2}$$

The relationship between the control input and the state vector is formulated in Section 5.2.1 by means of a motion model (also called the state transition model). The transformation between the measurement in robot coordinates and the map in world coordinates is given in Section 5.2.2.

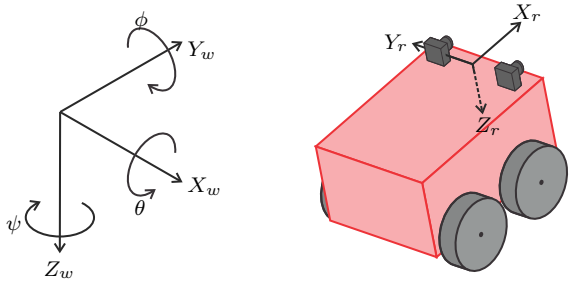



**Figure 5.1** – The robot axes moving with the robot in the stationary world axis frame.

### 5.2.1 Vehicle motion model

At every time step the controller of the robot generates a forward and angular velocity command, that is

$$\hat{\mathbf{u}}_t = \begin{bmatrix} v \\ \dot{\psi} \end{bmatrix}, \tag{5.2.3}$$

with $v$ the forward translational speed and $\dot{\psi}$ the angular velocity (about the $Z_r$ axis). The control input is used by the robot to control the rotational speed of the wheels.

Since we want to be able to describe motion in 3D, we extend the control input to include angular velocities about the the other two axes as

$$\mathbf{u}_t = \begin{bmatrix} v \\ \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{M}_t), \tag{5.2.4}$$

with $\dot{\theta}$ and $\dot{\phi}$ angular velocities about the $X$ axis and $Y$ axis respectively. Although $\dot{\theta}$ and $\dot{\phi}$ will always be zero for a terrestrial vehicle, we can model uncertainty in the $z_t$, $\theta_t$ and $\phi_t$ states by including them in the control input. Due to vehicle model variance and wheel slippage, the robot may not do exactly what we expect. To characterize this process noise, we add zero mean Gaussian noise with covariance matrix

$$\mathbf{M}_t = \begin{bmatrix} \alpha_1 v^2 + \alpha_2 \dot{\psi}^2 & 0 & 0 & 0 \\ 0 & \alpha_3 v^2 + \alpha_4 \dot{\psi}^2 & 0 & 0 \\ 0 & 0 & \alpha_5 v^2 + \alpha_6 \dot{\psi}^2 & 0 \\ 0 & 0 & 0 & \alpha_7 v^2 + \alpha_8 \dot{\psi}^2 \end{bmatrix}, \tag{5.2.5}$$

as is common practice in the field [34, p. 127]. The $\alpha$ parameters are robot and environment specific, and have to be estimated with practical testing and some degree of guesswork.

In order to update the robot states with the control input we define the motion equation as

$$\mathbf{x}_t = \mathbf{g}(\mathbf{x}_{t-1}, \mathbf{u}_t) = \begin{bmatrix} x_t \\ y_t \\ z_t \\ \theta_t \\ \phi_t \\ \psi_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \\ \theta_{t-1} \\ \phi_{t-1} \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{R}_{t-1} \begin{bmatrix} vT\cos(\dot{\psi}T) \\ vT\sin(\dot{\psi}T) \\ 0 \end{bmatrix} \\ \dot{\theta}T \\ \dot{\phi}T \\ \dot{\psi}T \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{S}_t), \tag{5.2.6}$$

with $T$ the sample period of the system. This equation is also modelled with zero mean Gaussian noise (with covariance $\mathbf{S}_t$). We linearize the transition from $\mathbf{M}_t$ to $\mathbf{S}_t$ with a first order Taylor approximation, using the Jacobian

$$\mathbf{V}_t = \begin{bmatrix} \frac{\partial x_t}{\partial v} & \frac{\partial x_t}{\partial \dot{\theta}} & \frac{\partial x_t}{\partial \dot{\phi}} & \frac{\partial x_t}{\partial \dot{\psi}} \\ \frac{\partial y_t}{\partial v} & \frac{\partial y_t}{\partial \dot{\theta}} & \frac{\partial y_t}{\partial \dot{\phi}} & \frac{\partial y_t}{\partial \dot{\psi}} \\ \frac{\partial z_t}{\partial v} & \frac{\partial z_t}{\partial \dot{\theta}} & \frac{\partial z_t}{\partial \dot{\phi}} & \frac{\partial z_t}{\partial \dot{\psi}} \\ \frac{\partial \theta_t}{\partial v} & \frac{\partial \theta_t}{\partial \dot{\theta}} & \frac{\partial \theta_t}{\partial \dot{\phi}} & \frac{\partial \theta_t}{\partial \dot{\psi}} \\ \frac{\partial \phi_t}{\partial v} & \frac{\partial \phi_t}{\partial \dot{\theta}} & \frac{\partial \phi_t}{\partial \dot{\phi}} & \frac{\partial \phi_t}{\partial \dot{\psi}} \\ \frac{\partial \psi_t}{\partial v} & \frac{\partial \psi_t}{\partial \dot{\theta}} & \frac{\partial \psi_t}{\partial \dot{\phi}} & \frac{\partial \psi_t}{\partial \dot{\psi}} \end{bmatrix}, \tag{5.2.7}$$

evaluated at the updated robot state vector. The transformation is then given by

$$\mathbf{S}_t = \mathbf{V}_t \mathbf{M}_t \mathbf{V}_t^T. \tag{5.2.8}$$

With Equation 5.2.6 we can now describe the motion of the robot with a given control input and use the transformation of Equation 5.2.8 to describe the process noise in the six states of the robot brought on by its motion.

### 5.2.2 Measurement model

Having already discussed the necessary equations to describe the measurement in robot coordinates in Section 4.2, we now proceed with a transformation to world coordinates. We denote the location of landmark $i$ in the map corresponding to measurement $j$ at time $t$ as

$$\mathbf{m}_{i,t} = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \quad \text{and} \quad \mathbf{z}_{j,t} = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} \tag{5.2.9}$$

respectively. The measurement $\mathbf{z}_{j,t}$ will always be as the robot observes the landmark in robot coordinates, and the landmark's location $\mathbf{m}_{i,t}$ will always be given in world coordinates. In the previous chapter we defined $\mathbf{Q}_j$ as the measurement noise covariance relating to measurement $\mathbf{z}_{j,t}$. The transformation between robot and world coordinates is given by the measurement equation

$$\mathbf{z}_{j,t} = \mathbf{h}(\mathbf{x}_t, \mathbf{m}_{i,t}) = \mathbf{R}_t^T \begin{bmatrix} x_w - x_t \\ y_w - y_t \\ z_w - z_t \end{bmatrix}, \tag{5.2.10}$$

or inversely,

$$\mathbf{m}_{i,t} = \mathbf{h}^{-1}(\mathbf{x}_t, \mathbf{z}_{j,t}) = \mathbf{R}_t \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}. \tag{5.2.11}$$

Exactly which measurement corresponds to which landmark in the map is stored in a correspondence vector $\mathbf{c}_t$.

With the motion and measurement models we are able to use all the information available to optimally estimate the location of the robot and the locations of the measured landmarks. Exactly how we do this estimation is the topic of the next three sections.

## 5.3 EKF SLAM

In this thesis we investigate the use of three SLAM algorithms. The first approach we follow to solve the SLAM problem is based on the extended Kalman filter (EKF).

The EKF is probably the most well-known filter used for state estimation. The EKF linearizes plant and measurement models about the current estimate to enable the use of normal Kalman filter equations. A more in-depth discussion of the EKF is given in Appendix A. The intuitive nature of the EKF makes it easy to use and understand, and therefore an obvious first method to try with the SLAM problem.

It is important to note that when we work with the EKF we operate with the assumption that the uncertainties in measurement and robot motion are Gaussian. We have taken care thus far to characterize the process and measurement noise to be Gaussian (and verified the Gaussian measurement assumption in Section 4.2).

To use the EKF for SLAM we need to define a vector of states to be estimated by the EKF. This vector includes the robot states as well as the locations of all the landmarks, and is of the form

$$\boldsymbol{\mu}_t = \begin{bmatrix} \mathbf{x}_t^T & \mathbf{m}_{1,t}^T & \cdots & \mathbf{m}_{n,t}^T \end{bmatrix}^T. \tag{5.3.1}$$

This vector has $6 + 3n$ elements, where $n$ is the number of landmarks being maintained. Corresponding to the mean vector we have the $(6 + 3n)\times(6 + 3n)$ covariance matrix $\boldsymbol{\Sigma}_t$.

The EKF SLAM algorithm as it is executed at every time step is given in Algorithm 5.1 (this algorithm is an adaptation of the one given in [34, p. 314]). The EKF can be divided into two parts: the control or time update (lines 1 to 4 in the algorithm) and the measurement update (lines 5 to 15). The control update uses the control input with the motion model to calculate an initial estimate of the robot's new location at every time step. Because this step is done without any measurement, we expect the uncertainty in robot states to increase. The measurement update uses all the measurements one by one to (ideally) decrease the uncertainty of both robot and landmark states.

---

**Algorithm 5.1** EKF_SLAM($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t, \mathbf{c}_t$)

1: $\mathbf{x}_t := \mathbf{g}(\mathbf{x}_{t-1}, \mathbf{u}_t)$
2: $\boldsymbol{\mu}_t := \begin{bmatrix} \mathbf{x}_t^T & \mathbf{m}_{1,t-1}^T & \cdots & \mathbf{m}_{n,t-1}^T \end{bmatrix}^T$
3: $\mathbf{G}_t := \mathbf{J}_{\mathbf{g}}(\mathbf{x}_{t-1})\mathbf{F}$
4: $\boldsymbol{\Sigma}_t := \mathbf{G}_t\boldsymbol{\Sigma}_{t-1}\mathbf{G}_t^T + \mathbf{S}_t$
5: **for** all observed landmarks $\mathbf{z}_{i,t}$ **do**
6: $\quad j := c_{i,t}$
7: $\quad$ **if** landmark $j$ had never been seen before **then**
8: $\quad\quad \mathbf{m}_{j,t} := \mathbf{h}^{-1}(\mathbf{x}_t, \mathbf{z}_{i,t})$
9: $\quad$ **end if**
10: $\quad \hat{\mathbf{z}} := \mathbf{h}(\mathbf{x}_t, \mathbf{m}_{j,t})$
11: $\quad \mathbf{H} := \mathbf{J}_{\mathbf{h}}(\mathbf{x}_t, \mathbf{m}_{j,t})\mathbf{F}_j$
12: $\quad \mathbf{K} := \boldsymbol{\Sigma}_t\mathbf{H}^T(\mathbf{H}\boldsymbol{\Sigma}_t\mathbf{H}^T + \mathbf{Q}_i)^{-1}$
13: $\quad \boldsymbol{\mu}_t := \boldsymbol{\mu}_t + \mathbf{K}(\mathbf{z}_{i,t} - \hat{\mathbf{z}})$
14: $\quad \boldsymbol{\Sigma}_t := (\mathbf{I} - \mathbf{K}\mathbf{H})\boldsymbol{\Sigma}_t$
15: **end for**
16: **return** $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

---

In order to clarify, we now look at the algorithm in detail by discussing each step.

- **Lines 1 and 2:** The first step of the EKF is to do the control update that estimates the state vector using the control input and the state estimate from the previous time step. This is executed by first updating the robot state vector using the function g given in Equation 5.2.6. An initial state vector is then compiled using the new robot states with the map states from the previous time step.

- **Lines 3 and 4:** In order to include the uncertainty in robot states brought on by the motion model, a linear approximation of the motion is used. The linearization is done with a first order Taylor approximation using the Jacobian matrix of the function g evaluated at the robot state

vector from the previous time step, that is

$$\mathbf{J_g}(\mathbf{x}_{t-1}) = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial x_t}{\partial y_{t-1}} & \frac{\partial x_t}{\partial z_{t-1}} & \frac{\partial x_t}{\partial \theta_{t-1}} & \frac{\partial x_t}{\partial \phi_{t-1}} & \frac{\partial x_t}{\partial \psi_{t-1}} \\ \frac{\partial y_t}{\partial x_{t-1}} & \frac{\partial y_t}{\partial y_{t-1}} & \frac{\partial y_t}{\partial z_{t-1}} & \frac{\partial y_t}{\partial \theta_{t-1}} & \frac{\partial y_t}{\partial \phi_{t-1}} & \frac{\partial y_t}{\partial \psi_{t-1}} \\ \frac{\partial z_t}{\partial x_{t-1}} & \frac{\partial z_t}{\partial y_{t-1}} & \frac{\partial z_t}{\partial z_{t-1}} & \frac{\partial z_t}{\partial \theta_{t-1}} & \frac{\partial z_t}{\partial \phi_{t-1}} & \frac{\partial z_t}{\partial \psi_{t-1}} \\ \frac{\partial \theta_t}{\partial x_{t-1}} & \frac{\partial \theta_t}{\partial y_{t-1}} & \frac{\partial \theta_t}{\partial z_{t-1}} & \frac{\partial \theta_t}{\partial \theta_{t-1}} & \frac{\partial \theta_t}{\partial \phi_{t-1}} & \frac{\partial \theta_t}{\partial \psi_{t-1}} \\ \frac{\partial \phi_t}{\partial x_{t-1}} & \frac{\partial \phi_t}{\partial y_{t-1}} & \frac{\partial \phi_t}{\partial z_{t-1}} & \frac{\partial \phi_t}{\partial \theta_{t-1}} & \frac{\partial \phi_t}{\partial \phi_{t-1}} & \frac{\partial \phi_t}{\partial \psi_{t-1}} \\ \frac{\partial \psi_t}{\partial x_{t-1}} & \frac{\partial \psi_t}{\partial y_{t-1}} & \frac{\partial \psi_t}{\partial z_{t-1}} & \frac{\partial \psi_t}{\partial \theta_{t-1}} & \frac{\partial \psi_t}{\partial \phi_{t-1}} & \frac{\partial \psi_t}{\partial \psi_{t-1}} \end{bmatrix}. \tag{5.3.2}$$

The matrix $\mathbf{F}$, given by

$$\mathbf{F} = \begin{bmatrix} \mathbf{I}_6 & \mathbf{O}_{6\times 3n} \end{bmatrix}, \tag{5.3.3}$$

is used to reshape the Jacobian matrix so that we can multiply it with the full state covariance matrix, as is done with the standard EKF equation in Line 4.

- **Lines 5 and 6:** We now consider every measured landmark separately. Within the loop, the index $j$ refers to the location of the current landmark in the state vector, while $i$ refers to the index of the landmark in the measurement as given by the correspondence vector $\mathbf{c}_t$.

- **Lines 7 to 9:** New landmarks that have not been seen before are included in the state vector with the inverse of the measurement function, given by Equation 5.2.11. Since we do not have any information about the location of a new landmark before it is measured, new landmarks are initiated with a theoretical uncertainty of infinity.

- **Line 10:** We calculate the location of the landmark in robot coordinates to be used with the update of the state vector by transforming the current state estimate of the landmark with the measurement function $\mathbf{h}$ given in Equation 5.2.10.

- **Line 11:** As with to the control update we linearize the measurement function with a Jacobian given by

$$\mathbf{J_h}(\mathbf{x}_t, \mathbf{m}_{j,t}) = \begin{bmatrix} \frac{\partial x_r}{\partial x_t} & \frac{\partial x_r}{\partial y_t} & \frac{\partial x_r}{\partial z_t} & \frac{\partial x_r}{\partial \theta_t} & \frac{\partial x_r}{\partial \phi_t} & \frac{\partial x_r}{\partial \psi_t} & \frac{\partial x_r}{\partial x_w} & \frac{\partial x_r}{\partial y_w} & \frac{\partial x_r}{\partial z_w} \\ \frac{\partial y_r}{\partial x_t} & \frac{\partial y_r}{\partial y_t} & \frac{\partial y_r}{\partial z_t} & \frac{\partial y_r}{\partial \theta_t} & \frac{\partial y_r}{\partial \phi_t} & \frac{\partial y_r}{\partial \psi_t} & \frac{\partial y_r}{\partial x_w} & \frac{\partial y_r}{\partial y_w} & \frac{\partial y_r}{\partial z_w} \\ \frac{\partial z_r}{\partial x_t} & \frac{\partial z_r}{\partial y_t} & \frac{\partial z_r}{\partial z_t} & \frac{\partial z_r}{\partial \theta_t} & \frac{\partial z_r}{\partial \phi_t} & \frac{\partial z_r}{\partial \psi_t} & \frac{\partial x_r}{\partial x_w} & \frac{\partial z_r}{\partial y_w} & \frac{\partial z_r}{\partial z_w} \end{bmatrix}. \tag{5.3.4}$$

We once again use a reshaping matrix $\mathbf{F}_j$ to have the dimensions of the Jacobian correspond to the full state covariance matrix. This reshaping matrix is given as

$$\mathbf{F}_j = \begin{bmatrix} \mathbf{I}_6 & \mathbf{O}_{6\times 3j-3} & \mathbf{O}_{6\times 3} & \mathbf{O}_{6\times 3n-3j} \\ \mathbf{O}_{3\times 6} & \mathbf{O}_{6\times 3j-3} & \mathbf{I}_3 & \mathbf{O}_{3\times 3n-3j} \end{bmatrix}. \tag{5.3.5}$$

- **Lines 12 to 14:** This part of the algorithm utilizes the standard EKF equations to calculate the Kalman gain $\mathbf{K}$, and uses it to update the state vector with the error between measurement estimate and measurement. The covariance matrix is also updated with the Kalman gain and the Jacobian of the measurement equation.

- **Line 16** Finally the new state vector and covariance matrix is returned to be used at the next time step.

Figure 5.2 illustrates the working of the EKF SLAM algorithm. The robot starts with no pose uncertainty and observes some landmarks. At every consecutive time step, the uncertainty in robot

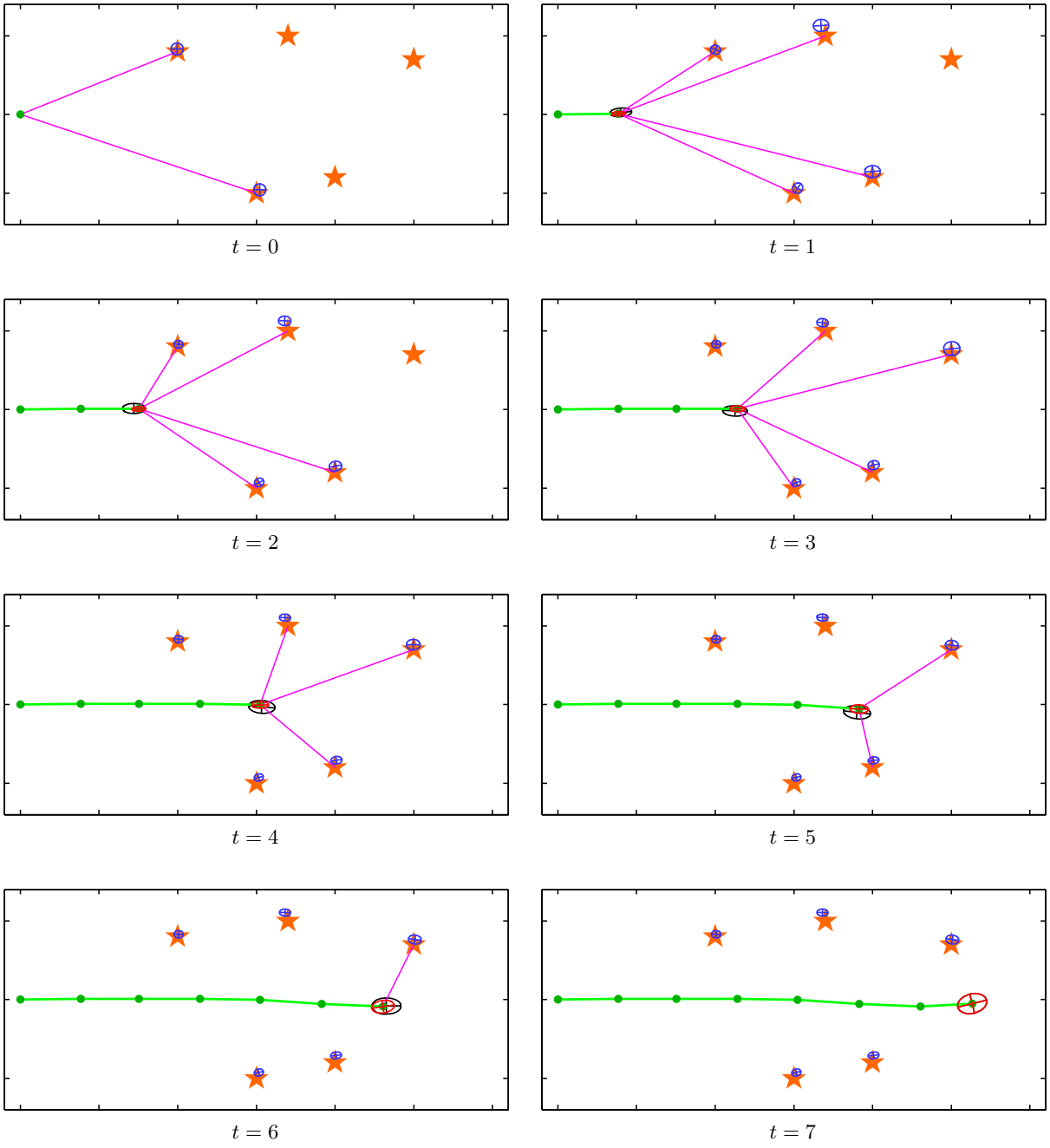



**Figure 5.2** – A step by step execution of EKF SLAM. The robot's estimated route is given in green, the true landmark locations as orange stars and the observation at every time step as magenta lines. Standard deviation confidence ellipses indicate the Gaussian mean and covariance in states. Black ellipses represent uncertainty after the control update, red uncertainty after the measurement update and blue the landmark estimates.

location is increased by the control update, and decreased by the measurement update. This saw-tooth behaviour of uncertainty is typical of the EKF. The uncertainty corresponding to a landmark in the map also decreases as the landmark is observed multiple times.

From this explanation of the algorithm two problems can be highlighted: the algorithm can become computationally expensive when a large map is used, and it is unable to model non-Gaussian uncertainty in the state estimate. A third major problem concerns data association, and in the next chapter we show just how catastrophic incorrect landmark matches can be. In an attempt to overcome these limitations of the EKF, we next look at two particle filter-based SLAM algorithms, namely FastSLAM and FastSLAM 2.

## 5.4 FastSLAM

Since the particle filter can be used to approximate any distribution, it is often utilized to estimate non-Gaussian systems [34, p. 96]. Similar to the EKF the particle filter uses a motion model to predict the state estimate at every time step, and measurements to refine this estimate. Instead of using a Gaussian distribution to describe the state estimate, the particle filter uses a distribution of sample points (called particles). These particles can describe any probability function, with a degree of accuracy depending on the number of particles used.

At every time step particles are propagated using the motion model and its noise model. Particles are then weighted according to how well they correspond with the measurement. If the set of particles contains a number of particles with very low weights, the set is resampled by copying strong particles and letting weak particles die. A more in-depth explanation of the particle filter is given in Appendix A.

A major drawback of the particle filter, however, is that with high-dimensional problems a large number of particles are needed to describe the distribution adequately. While we are already expecting computational problems with the EKF, which scales quadratically with number of dimensions, we now find that the problem is exacerbated by the particle filter, as it scales exponentially. The Rao-Blackwellized particle filter [10] was developed in an effort to overcome this problem. This filter uses particles to describe some states and Gaussian distributions to represent all other states. In order to utilize it, we factorize the SLAM problem given in Equation 5.1.1 as

$$p(\mathbf{x}_t, \mathbf{m} \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) = p(\mathbf{x}_t \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \prod_{i=1}^{n} p(\mathbf{m}_i \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t}). \tag{5.4.1}$$

With this factorization we describe the posterior as a product of $n+1$ probabilities. If we suppose the exact location of the robot is known, the landmark positions will be independent from one another and can therefore be estimated independently. Naturally we do not know the robot's location, but this independence can be utilized when using particles to estimate the robot position (a mathematical derivation is provided in [34, p. 442]). It can even be shown that the above factorization is exact and not an approximation [26].

FastSLAM uses a particle filter to compute the posterior $p(\mathbf{x}_t \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$ over robot states, and a separate EKF for every landmark in the map to obtain $p(\mathbf{m}_i \,|\, \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$. What this means is that instead of only one filter, we factor the problem into $1 + nm$ filters, where $m$ is the number of particles. The large number of filters may seem excessive, but because of the low dimensionality of every filter the algorithm is remarkably efficient.

We define every particle to have a state vector for the robot states, and a mean vector and covariance matrix for every landmark, as

$$Y_t^{[k]} = \left\langle \mathbf{x}_t^{[k]}, \langle \mathbf{m}_{1,t}^{[k]}, \mathbf{\Sigma}_{1,t}^{[k]} \rangle, \ldots, \langle \mathbf{m}_{n,t}^{[k]}, \mathbf{\Sigma}_{n,t}^{[k]} \rangle \right\rangle, \tag{5.4.2}$$

with $\mathbf{x}_t^{[k]}$ the robot location and orientation for particle $k$, and $\left\langle \mathbf{m}_{i,t}^{[k]}, \mathbf{\Sigma}_{i,t}^{[k]} \right\rangle$ the $i$-th landmark's Gaussian mean and covariance. The FastSLAM algorithm is given below in Algorithm 5.2, and we proceed with a step by step explanation.

---

**Algorithm 5.2** FastSLAM($Y_{t-1}, \mathbf{u}_t, \mathbf{z}_t, \mathbf{c}_t$)

**for** all particles $k \in \{1, 2, \ldots, m\}$ **do**
  $\mathbf{x}_t^{[k]} \sim p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[k]}, \mathbf{u}_t)$
  **for** all observed landmarks $\mathbf{z}_{i,t}$ **do**
    $j := c_{i,t}$
    **if** landmark $j$ had never been seen before **then**
      $\mathbf{m}_{j,t}^{[k]} = \mathbf{h}^{-1}(\mathbf{x}_t^{[k]}, \mathbf{z}_{i,t})$
      $\mathbf{H}_j := \mathbf{J}_\mathbf{h}(\mathbf{m}_{j,t}^{[k]})$
      $\mathbf{\Sigma}_{j,t}^{[k]} := (\mathbf{H}_j^{-1})\mathbf{Q}_i(\mathbf{H}_j^{-1})^T$
    **else**
      $\hat{\mathbf{z}} := \mathbf{h}(\mathbf{x}_t^{[k]}, \mathbf{m}_{j,t}^{[k]})$
      $\mathbf{H}_j := \mathbf{J}_\mathbf{h}(\mathbf{m}_{j,t}^{[k]})$
      $\mathbf{Q} := \mathbf{H}\mathbf{\Sigma}_{j,t-1}^{[k]}\mathbf{H}^T + \mathbf{Q}_i$
      $\mathbf{K} := \mathbf{\Sigma}_{j,t-1}^{[k]}\mathbf{H}_j^T\mathbf{Q}^{-1}$
      $\mathbf{m}_{j,t}^{[k]} := \mathbf{m}_{j,t-1}^{[k]} + \mathbf{K}(\mathbf{z}_{i,t} - \hat{\mathbf{z}})$
      $\mathbf{\Sigma}_{j,t}^{[k]} := (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{\Sigma}_{j,t-1}^{[k]}$
      $w^{[k]} := w^{[k]} f(\mathbf{Q}, \mathbf{z}_{i,t}, \hat{\mathbf{z}})$
    **end if**
  **end for**
  **for** all other landmarks $j' \neq \mathbf{c}_t$ **do**
    $\mathbf{m}_{j',t}^{[k]} := \mathbf{m}_{j',t-1}^{[k]}$
    $\mathbf{\Sigma}_{j',t}^{[k]} := \mathbf{\Sigma}_{j',t-1}^{[k]}$
  **end for**
**end for**
**for** all $k \in \{1, 2, \ldots, m\}$ **do**
  draw random particle $k$ with probability $\propto w^{[k]}$
  include $\left\langle \mathbf{x}_t^{[k]}, \langle \mathbf{m}_{1,t}^{[k]}, \mathbf{\Sigma}_{1,t}^{[k]} \rangle, \ldots, \langle \mathbf{m}_{n,t}^{[k]}, \mathbf{\Sigma}_{n,t}^{[k]} \rangle \right\rangle$ in $Y_t$
**end for**
**return** $Y_t$

---

- **Lines 1 and 2:** As with a normal particle filter, the FastSLAM algorithm begins by entering a loop over all particles. The control input is used to sample a new robot pose for every particle according to the uncertainty in the motion model. We add random noise drawn from a zero mean Gaussian distribution with a covariance of $\mathbf{M}_t$, given by Equation 5.2.5, to the control input and use the motion equation $\mathbf{g}$, given by Equation 5.2.6, to find the new location and orientation of each particle. This distribution from which we draw particles is often referred to as the proposal distribution.

- **Lines 3 and 4:** For every particle we enter a loop over all the measured landmarks. For every iteration the algorithm does one of two things: it adds a new landmark, or updates an old landmark. The index in the map of an old landmark is given by the correspondence vector $\mathbf{c}_t$.

- **Lines 5 to 8:** A new landmark is added to the map using the measurement equation $\mathbf{h}$ to calculate its location in world coordinates. Since we want to use an EKF to estimate each landmark, we have to linearize the measurement model by using a first order Taylor approximation with the Jacobian

$$\mathbf{J}_{\mathbf{h}}(\mathbf{x}_t, \mathbf{m}_{j,t}) = \begin{bmatrix} \frac{\partial x_r}{\partial x_w} & \frac{\partial x_r}{\partial y_w} & \frac{\partial x_r}{\partial z_w} \\ \frac{\partial y_r}{\partial x_w} & \frac{\partial y_r}{\partial y_w} & \frac{\partial y_r}{\partial z_w} \\ \frac{\partial z_r}{\partial x_w} & \frac{\partial z_r}{\partial y_w} & \frac{\partial z_r}{\partial z_w} \end{bmatrix}. \tag{5.4.3}$$

  With this Jacobian we transform the uncertainty in measurement to an uncertainty in world coordinates. Unlike EKF SLAM, the location of the robot holds no uncertainty to be added to the measurement uncertainty. The uncertainty of the robot location is encapsulated in the distribution of the particles.

- **Lines 9 to 15:** If a landmark has been observed before, we use the normal EKF equations to update its state vector and covariance. The state estimate is calculated using the measurement model which is linearized with the same Jacobian that we used for new landmarks.

- **Line 16:** Once the landmark has been updated by using the measurement we have to calculate its effect on the weighting of the particle in question. As with a normal particle filter the importance weight is calculated as the probability of the measurement given the particle in question, so that

$$w^{[k]} = \frac{\text{target distribution}}{\text{proposal distribution}} \tag{5.4.4}$$

$$= \frac{p(\mathbf{x}_{1:t}^{[k]} \mid \mathbf{z}_{1:t},\, \mathbf{u}_{1:t})}{p(\mathbf{x}_{1:t}^{[k]} \mid \mathbf{z}_{1:t-1},\, \mathbf{u}_{1:t})}. \tag{5.4.5}$$

  The weighting function used in the algorithm can be shown [34, p. 448] to be

$$f(\mathbf{Q}, \mathbf{z}_{i,t}, \hat{\mathbf{z}}) = |\mathbf{Q}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{z}_{i,t}-\hat{\mathbf{z}})^T \mathbf{Q}^{-1}(\mathbf{z}_{i,t}-\hat{\mathbf{z}})}. \tag{5.4.6}$$

  It is not necessary to update the weights for new landmarks as those weights will be the same for all particles, and therefore have no overall effect when we normalize the weights.

- **Lines 19 to 22:** If a previously observed feature has not been observed at the current time step its state vector and uncertainty remain unchanged. All unobserved landmarks are therefore essentially ignored. This property of the algorithm is especially useful when a large map is maintained as the number of unseen landmarks in the map do not impact the execution time.

- **Lines 24 to 27:** Resampling is done by drawing particles with probabilities proportional to their normalized weights. Particles with a low weight will be more likely to perish while particles with a high weight will be copied and used at the next time step.

- **Line 28:** Finally, the updated and resampled particles are returned to be used at the next time step.

The importance of resampling is depicted in Figure 5.3. Without resampling the particles cannot describe the posterior accurately and will simply keep on spreading out until only one particle remains in any way relevant. This phenomenon is called particle depreciation. A disadvantage of resampling is that, over time, landmarks that have been observed in the past will be described by fewer unique particles until ultimately (and inevitably) only one particle from that time will still be active.

Figure 5.4 depicts the execution of FastSLAM on the same small simulation we use with EKF SLAM in Figure 5.2. All particles start at the same location where two landmarks are observed. At the next time step the particles are propagated by using the motion model and associated uncertainty. For every particle the location of the robot state and landmark means are shown with their weighting at that time step.

A powerful possibility created by the use of particles is that of multiple hypothesis tracking. What this would entail is that particles would represent possible paths that the robot could have taken, and we can even calculate landmark correspondences for every particle separately. Because of the expensive nature of calculating feature matches, we decide against this procedure and, instead, calculate one correspondence vector for all the particles. It is, however, important to note that the algorithm creates this possibility and future extensions can explore this feature.

If we work in a 3D environment, the FastSLAM algorithm requires a large number of particles to accurately describe the posterior. When we consider Figure 5.4, it is clear that only a fraction of the particles contain relevant information and survive the resampling while a large number have near zero weights. The next section explores the possibility of refining the distribution from which particles are sampled, thereby enabling us to reduce the number of particles that are needed without losing accuracy.

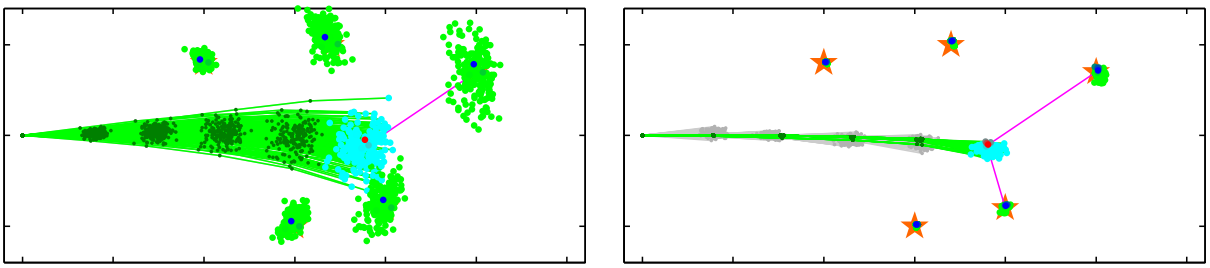


**Figure 5.3** – The importance of resampling. Without resampling most particles become meaningless with near zero weight, as is shown on the left. With resampling the posterior can be described more accurately over a long time.

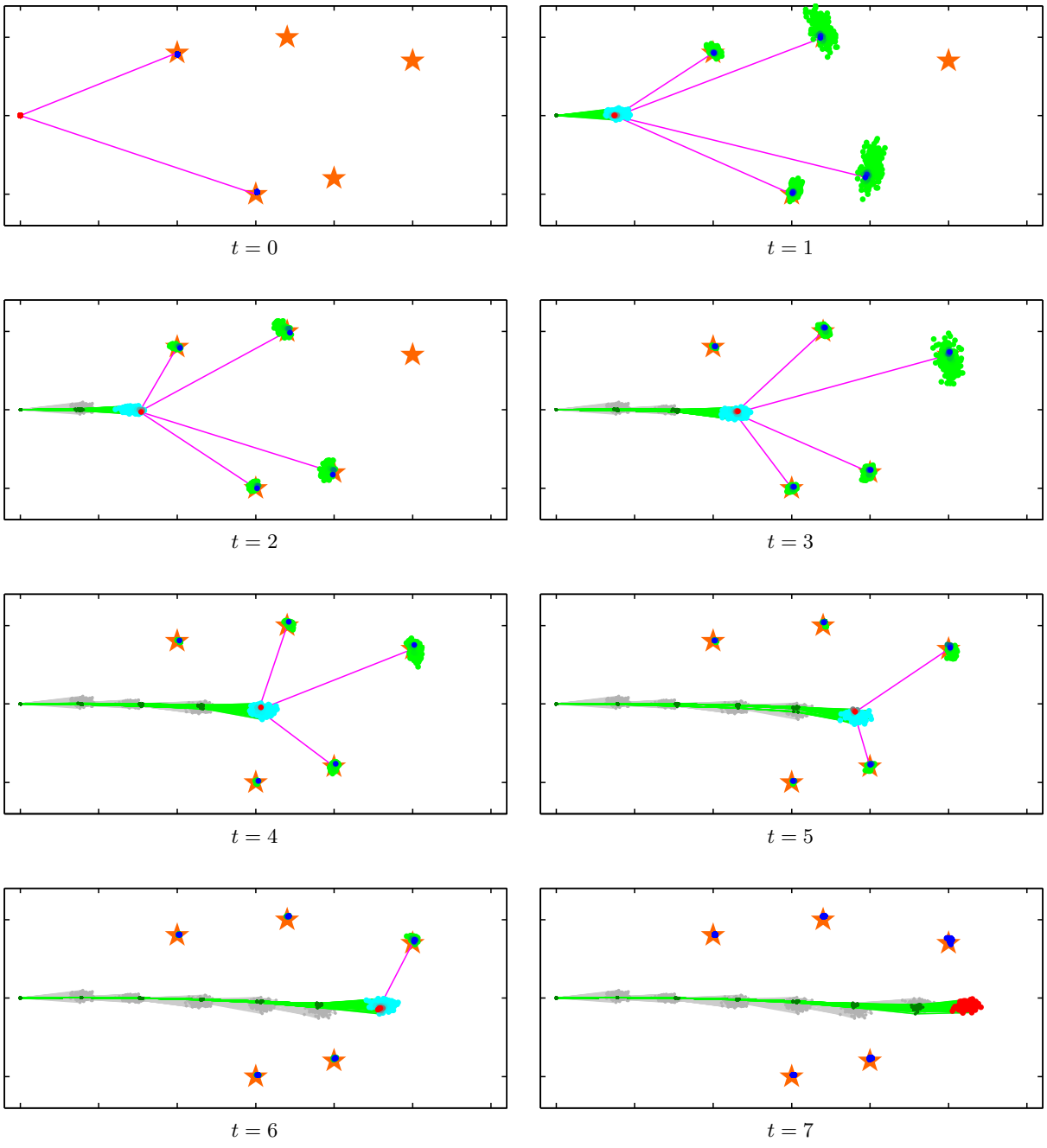



**Figure 5.4** – A step by step execution of FastSLAM. The particles start at the same location at the first time step. At consecutive time steps the route estimated by the particles is shown in green, with the current estimate scaled between red and cyan (cyan for a weight of zero and red for a normalized weight of one). Similarly, the mean of the landmark estimates for every particle is shown, scaled between green (low) and blue (high) according to particle weights. The true locations of landmarks are depicted with orange stars and measurements with magenta lines. Particles that have not survived the resampling are shown in grey.

## 5.5 FastSLAM 2

FastSLAM and FastSLAM 2 are in many ways similar. Both algorithms use the Rao-Blackwellized particle filter structure with particles describing the pose of the robot and an EKF for every landmark in the map of every particle. The main difference is that instead of sampling from the distribution

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[k]}, \mathbf{u}_t), \tag{5.5.1}$$

FastSLAM 2 calculates a new proposal distribution

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}^{[k]}, \mathbf{u}_t, \mathbf{z}_t), \tag{5.5.2}$$

and samples particles from it. The difference may seem minor, but by including the measurement the mean and covariance of the proposal distribution can be refined substantially. This extension does, however, complicate the implementation somewhat.

In order to calculate the proposal distribution, the algorithm linearizes the motion and measurement equations, similar to the way in which it was done with the EKF SLAM algorithm. The FastSLAM 2 algorithm is given in Algorithm 5.3, and we proceed with a step by step explanation.

- **Line 1:** The algorithm starts as most particle filters do by entering a loop over all the particles.
- **Lines 2 and 3:** An initial proposal distribution is calculated with the motion model. It is from this normal distribution, with mean $\hat{\mathbf{x}}_t$ and covariance $\mathbf{S}_t$, that FastSLAM 2 samples its particles.
- **Lines 4 to 12:** The algorithm now enters a loop over all the measured features. With every measurement the proposal distribution is refined by using an EKF style approximation of the measurement model. This approximation is necessary if Equation 5.5.2 is to have a closed form solution. We use two Jacobians for this approximation,

$$\mathbf{H}_{j,x} = \mathbf{J}_{\mathbf{h}}(\hat{\mathbf{x}}_t, \mathbf{m}_{j,t}) = \begin{bmatrix} \frac{\partial x_r}{\partial x_t} & \frac{\partial x_r}{\partial y_t} & \frac{\partial x_r}{\partial z_t} & \frac{\partial x_r}{\partial \theta_t} & \frac{\partial x_r}{\partial \phi_t} & \frac{\partial x_r}{\partial \psi_t} \\ \frac{\partial y_r}{\partial x_t} & \frac{\partial y_r}{\partial y_t} & \frac{\partial y_r}{\partial z_t} & \frac{\partial y_r}{\partial \theta_t} & \frac{\partial y_r}{\partial \phi_t} & \frac{\partial y_r}{\partial \psi_t} \\ \frac{\partial z_r}{\partial x_t} & \frac{\partial z_r}{\partial y_t} & \frac{\partial z_r}{\partial z_t} & \frac{\partial z_r}{\partial \theta_t} & \frac{\partial z_r}{\partial \phi_t} & \frac{\partial z_r}{\partial \psi_t} \end{bmatrix} \tag{5.5.3}$$

  and

$$\mathbf{H}_{j,m} = \mathbf{J}_{\mathbf{h}}(\mathbf{m}_{j,t}) = \begin{bmatrix} \frac{\partial x_r}{\partial x_w} & \frac{\partial x_r}{\partial y_w} & \frac{\partial x_r}{\partial z_w} \\ \frac{\partial y_r}{\partial x_w} & \frac{\partial y_r}{\partial y_w} & \frac{\partial y_r}{\partial z_w} \\ \frac{\partial z_r}{\partial x_w} & \frac{\partial z_r}{\partial y_w} & \frac{\partial z_r}{\partial z_w} \end{bmatrix}. \tag{5.5.4}$$

  With these Jacobians and the estimate of the measurement, $\hat{\mathbf{z}}$, the proposal distribution is refined by using the equations shown. Figure 5.5 depicts an example of this refinement.
- **Line 13:** Once the proposal distribution has been calculated, a particle is drawn from it.
- **Line 14 to 36:** The algorithm now proceeds almost exactly as FastSLAM does. The estimation of the landmarks in the map, the unseen landmarks, and the resampling are done exactly as before. The weighting, however, is different because of the new proposal distribution. The new weighting function is [34, p. 455]

$$f(\mathbf{Q}, \mathbf{z}_{i,t}, \hat{\mathbf{z}}) = |\mathbf{L}_i|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{z}_{i,t}-\hat{\mathbf{z}})^T \mathbf{L}_i^{-1}(\mathbf{z}_{i,t}-\hat{\mathbf{z}})}, \tag{5.5.5}$$

  with

$$\mathbf{L}_i = \mathbf{H}_{j,x}\mathbf{S}_t^{[k]}\mathbf{H}_{j,x}^T + \mathbf{H}_{j,m}\mathbf{\Sigma}_{j,t}^{[k]}\mathbf{H}_{j,m}^T + \mathbf{Q}. \tag{5.5.6}$$

**Algorithm 5.3** FastSLAM_2($Y_{t-1}, \mathbf{u}_t, \mathbf{z}_t, \mathbf{c}_t$)

**for** all particles $k \in \{1, 2, \ldots, m\}$ **do**
  $\hat{\mathbf{x}}_t^{[k]} := \mathbf{g}(\mathbf{x}_{t-1}^{[k]}, \mathbf{u}_t)$
  $\mathbf{S}_t^{[k]} := \mathbf{S}_t$
  **for** all observed landmarks $\mathbf{z}_{i,t}$ **do**
    $j := c_{i,t}$
    $\hat{\mathbf{z}} := \mathbf{h}(\mathbf{x}_t^{[k]}, \mathbf{m}_{j,t}^{[k]})$
    $\mathbf{H}_{j,x} := \mathbf{J}_\mathbf{h}(\hat{\mathbf{x}}_t^{[k]}, \mathbf{m}_{j,t}^{[k]})$
    $\mathbf{H}_{j,m} := \mathbf{J}_\mathbf{h}(\mathbf{m}_{j,t}^{[k]})$
    $\mathbf{Q}_i^{[k]} := \mathbf{Q}_i + \mathbf{H}_{j,m}\boldsymbol{\Sigma}_{j,t-1}^{[k]}\mathbf{H}_{j,m}^T$
    $\mathbf{S}_t^{[k]} := \left[\mathbf{H}_{j,x}^T \mathbf{Q}_i^{[k]\,-1} \mathbf{H}_{j,x} + \mathbf{S}_t^{[k]\,-1}\right]^{-1}$
    $\hat{\mathbf{x}}_t^{[k]} := \mathbf{S}_t^{[k]}\mathbf{H}_{j,x}^T \mathbf{Q}_i^{[k]\,-1}(\mathbf{z}_{i,t} - \hat{\mathbf{z}}) + \hat{\mathbf{x}}_t^{[k]}$
  **end for**
  $\mathbf{x}_t^{[k]} \sim \mathcal{N}(\hat{\mathbf{x}}_t^{[k]}, \mathbf{S}_t^{[k]})$
  **for** all observed landmarks $\mathbf{z}_{i,t}$ **do**
    $j := c_{i,t}$
    **if** landmark $j$ had never been seen before **then**
      $\mathbf{m}_{j,t}^{[k]} := \mathbf{h}^{-1}(\mathbf{x}_t^{[k]}, \mathbf{z}_{i,t})$
      $\mathbf{H}_{j,m} := \mathbf{J}_\mathbf{h}(\mathbf{m}_{j,t}^{[k]})$
      $\boldsymbol{\Sigma}_{j,t}^{[k]} := (\mathbf{H}_j^{-1})\mathbf{Q}_i(\mathbf{H}_j^{-1})^T$
    **else**
      $\hat{\mathbf{z}} := \mathbf{h}(\mathbf{x}_t^{[k]}, \mathbf{m}_{j,t}^{[k]})$
      $\mathbf{H}_{j,m} := \mathbf{J}_\mathbf{h}(\mathbf{m}_{j,t}^{[k]})$
      $\mathbf{H}_{j,x} := \mathbf{J}_\mathbf{h}(\hat{\mathbf{x}}_t^{[k]}, \mathbf{m}_{j,t}^{[k]})$
      $\mathbf{Q} := \mathbf{H}\boldsymbol{\Sigma}_{j,t-1}^{[k]}\mathbf{H}^T + \mathbf{Q}_i$
      $\mathbf{K} := \boldsymbol{\Sigma}_{j,t-1}^{[k]}\mathbf{H}_j^T\mathbf{Q}^{-1}$
      $\mathbf{m}_{j,t}^{[k]} := \mathbf{m}_{j,t-1}^{[k]} + \mathbf{K}(\mathbf{z}_{i,t} - \hat{\mathbf{z}})$
      $\boldsymbol{\Sigma}_{j,t}^{[k]} := (\mathbf{I} - \mathbf{K}\mathbf{H})\boldsymbol{\Sigma}_{j,t-1}^{[k]}$
      $w^{[k]} := w^{[k]} f(\mathbf{Q}, \mathbf{z}_{i,t}, \hat{\mathbf{z}})$
    **end if**
  **end for**
  **for** all other landmarks $j' \neq \mathbf{c}_t$ **do**
    $\mathbf{m}_{j',t}^{[k]} := \mathbf{m}_{j',t-1}^{[k]}$
    $\boldsymbol{\Sigma}_{j',t}^{[k]} := \boldsymbol{\Sigma}_{j',t-1}^{[k]}$
  **end for**
**end for**
Resample using $w^{[k]}$ to build a new $Y_t$
**return** $Y_t$

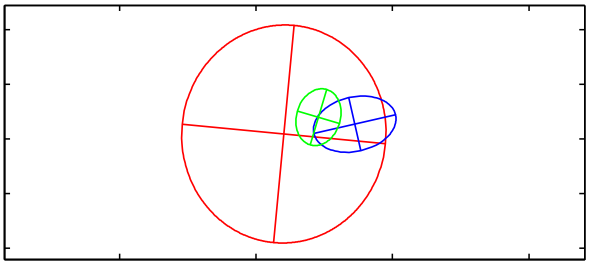


**Figure 5.5** – Refinement of the proposal distribution. The initial Gaussian distribution is depicted with the red confidence ellipse. By using the first measurement the uncertainty is decreased, shown in blue, and with the second measurement the distribution is refined even further, shown in green.

Because of the refined proposal distribution, FastSLAM 2 can be used accurately with significantly fewer particles than FastSLAM. It has been shown to give accurate results even with only one particle [26].

We once again employ the same small simulation to demonstrate the algorithm, as shown in Figure 5.6. Note that the particles are much closer together, and that there are far fewer insignificant particles, compared to FastSLAM shown in Figure 5.4. Also, far fewer particles die in the resampling process. This has the benefit of an accurate description of landmark uncertainty for a much longer time.

Now that we have described three SLAM algorithms (EKF SLAM, FastSLAM and FastSLAM 2), we can use them with realistic simulations to test assumptions and analyse performance in a controlled environment, before implementing them on a practical system. The next chapter describes the simulation environment that we created, as well as the performance of each of the algorithms in this environment.

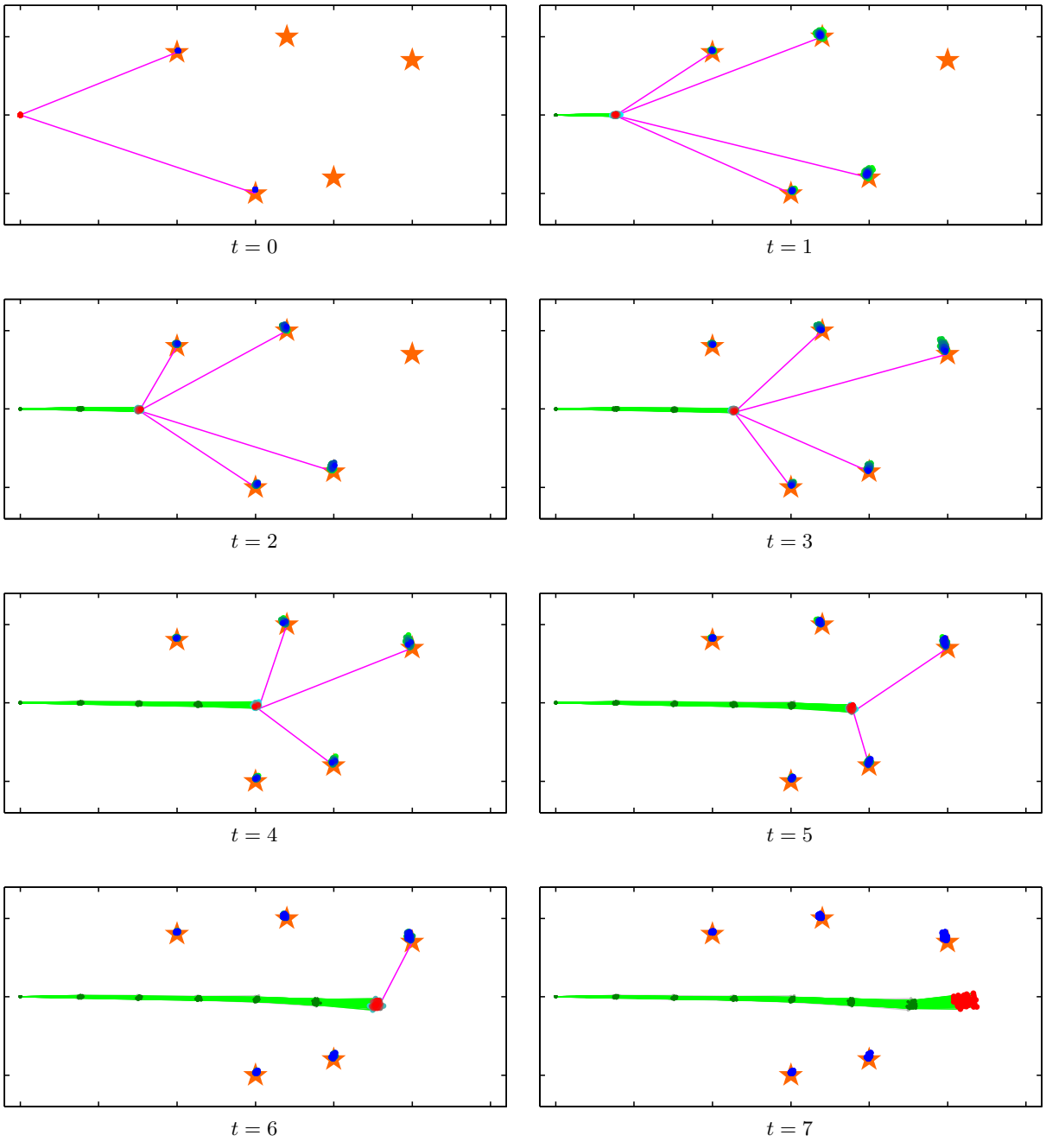



**Figure 5.6** – A step by step execution of FastSLAM 2. The particles start at the same location at the first time step. At consecutive time steps the route estimated by the particles is shown in green, with the current estimate scaled between red and cyan (cyan for weight of zero and red for a normalized weight of one). Similarly the mean of the landmark estimates for every particle is shown, scaled between green (low) and blue (high) according to particle weights. The true locations of landmarks are depicted with orange stars and measurements with magenta lines. Particles that have not survived the resampling are shown in grey. Note that the distribution of particles is smaller than we had with FastSLAM, and that there are far fewer irrelevant particles that do not survive the resampling.

# Chapter 6

# SLAM simulations

The aim of this chapter is to bring the stereo vision and SLAM theory together through a series of tests in a controlled simulation environment. The simulations are designed to emulate a mobile robot with a noisy control input and a measurement containing image coordinates of features on two images, similar to what we expect the output of a feature detector to be. Our simulation framework enables us to verify most of the theory covered in the previous three chapters and to evaluate our approaches before testing them on real world data.

We begin the chapter with a description of the simulation environment, and proceed with some implementation details and problems we encountered with regards to the SLAM algorithms. We provide results from several tests using these implementations, as well as from some situations that are likely to occur such as errors in feature matches. The chapter concludes with a discussion of the results.

## 6.1 Implementation

In order to test our SLAM systems we create two environments: one in 2D and one in 3D. These environments are created to provide a realistic representation of the real world and at the same time facilitate quantitative evaluation of performance. All three SLAM systems are implemented using the frameworks established in the previous chapter.

### 6.1.1 Simulation environment

We create our environment with the aim of simulating the real world without it being unnecessarily complicated. We opt for a route through a corridor-like environment with landmarks on the walls. Although these landmarks are more structured than they typically would be in a real world situation, the structure should not influence the result significantly and should have the benefit of being easy to evaluate visually. The 2D and 3D environments are essentially the same, except that the 3D environment also contains several ramps on which the robot will have nonzero roll or pitch angles.

In order to create a control input we supply waypoints for the simulated robot to follow. At each time step a simple gain controller generates an input command that steers it towards the next waypoint. This control input is stored for use in the SLAM simulations but, before the robot executes the command, we add some Gaussian noise to simulate the uncertainty that we know exists in this process (in other words, we add process noise to the control input). The robot's actual motion due to the noisy control is used as ground truth and to generate the measurements.

As the robot moves through the environment, landmarks in the robot's field of view are included in the measurement of every time step. Because feature detectors will sometimes see a landmark at one time step and not at the next, even if it is in the field of view, we add a probability that a landmark will be seen. We project the landmarks onto the image planes of two cameras fixed on the robot and then add Gaussian noise to the pixel coordinates. This noise model is in accordance with the model explained in Section 4.2. Each landmark is assigned a unique scalar to be used as a descriptor. By changing or mixing these descriptors in a measurement we can simulate feature mismatches and investigate their effect on the accuracy of the SLAM systems.

The 2D and 3D environments are depicted in Figures 6.2 (a) and 6.8 (a) with ground truth routes as well as routes calculated from using only the noisy control data.

### 6.1.2 SLAM algorithms

By using the algorithms given in the previous chapter we can implement the three SLAM systems: EKF SLAM, FastSLAM and FastSLAM 2. There are, however, some practicalities that we need to address.

The size of the EKF, or the number of landmarks we maintain in the state vector, has a quadratic effect on the execution time of the filter and must be selected carefully. With a few simple tests we find that as long as the filter can maintain a larger number than the largest measurement, accuracy will not be affected by this parameter. We choose the largest measurement that the system will have to handle, and make that the size of the map. An alternative to this approach would be to use a dynamic map size, but that can be memory inefficient. The replacement of landmarks in the filter is done by simply initiating a new landmark in the place of an old one. We mentioned in the previous chapter that a new landmark will initially have an uncertainty of infinity. In practice this is achieved by making the uncertainty very large, for example through a diagonal covariance with a standard deviation of a kilometre.

We find that the EKF SLAM algorithm is prone to instability (Thrun et al. also remark on this issue [34, p. 329]). To keep the filter stable we need to inflate the measurement noise somewhat. As long as the ratio between control and measurement noise does not change significantly it should not have a large effect on the estimated mean vector.

In the case of EKF SLAM we calculate one large covariance matrix for the entire system but, in order to visualize the uncertainty in landmark locations, we display confidence ellipses (or ellipsoids in 3D) corresponding to every landmark. We obtain separate covariances by marginalization, which amounts to extracting the relevant sub-matrices from the large covariance matrix [4, p. 89].

One of the benefits of the FastSLAM algorithms is that the size of the map has virtually no effect on the execution time of the algorithm. We take advantage of this by initiating the filter with a map of the same size as the number of landmarks present in the simulation. In the real world, however, we will be unable to accurately identify features that have not been seen for a while. To make our simulation more realistic we view landmarks that have not been observed for more that a certain number of time steps as new landmarks if they are observed again.

When we display a route estimated by one of the FastSLAM algorithms, we use a weighted average of the particles at every time step. An alternative here would be to use only the particle with the largest weight at every time step.

## 6.2 Simulation results

With the environment and the SLAM algorithms in place, we perform several tests and the results are given in this section. Note that by changing a few parameters in the simulation, such as the number of landmarks in the map and the measurement noise, the accuracy of the simulations can change significantly. Therefore, in the interest of fairness, we compare the SLAM algorithms using exactly the same inputs.

### 6.2.1 2D SLAM

We begin by evaluating all three algorithms in the 2D environment. At every time step each landmark has a 40% chance of being observed, but if it is observed, matching is done without error. We use the same measurements for all three algorithms. The results are depicted in Figures 6.1 and 6.2. Accuracy is measured by using the Euclidean error of the location estimate over time. Since the robot moves with a near constant forward velocity, the time and distance are proportional and the error over distance travelled will be similar to what we show. We find that the algorithms perform well despite noisy control and measurement inputs, and they produce remarkably similar results in terms of route and map accuracy.

We notice a degree of drift with all three algorithms. It is important to note that any system without an absolute location measurement (like GPS) will inevitably introduce drift. What we attempt is to limit the drift as much as possible.

In order to further quantify and compare the accuracy of the SLAM algorithms we run another three simulations, each with different conditions. Test 1 is done with very little noise and a large number of landmarks (an average of 32 per measurement). The noise conditions of this test is better than what we expect from the real world. For test 2 we use the noise parameters we expect from real world data and an average of 16 landmarks per measurement. Finally, test 3 is done with very few landmarks and noise parameters higher than what we expect. FastSLAM is executed with 250 particles and FastSLAM 2 with 80.

The results of these tests are summarized in Table 6.1. Since we work in MATLAB the average execution times per step that we provide should be viewed as unreliable, but we do observe that with optimized implementations the algorithms should be able to run in real time.

We calculate the percentage error over the total distance by dividing the Euclidean error at the

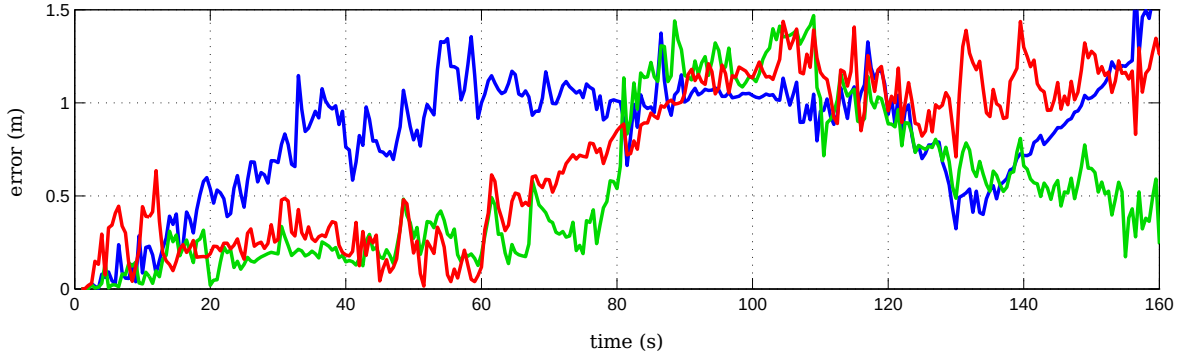
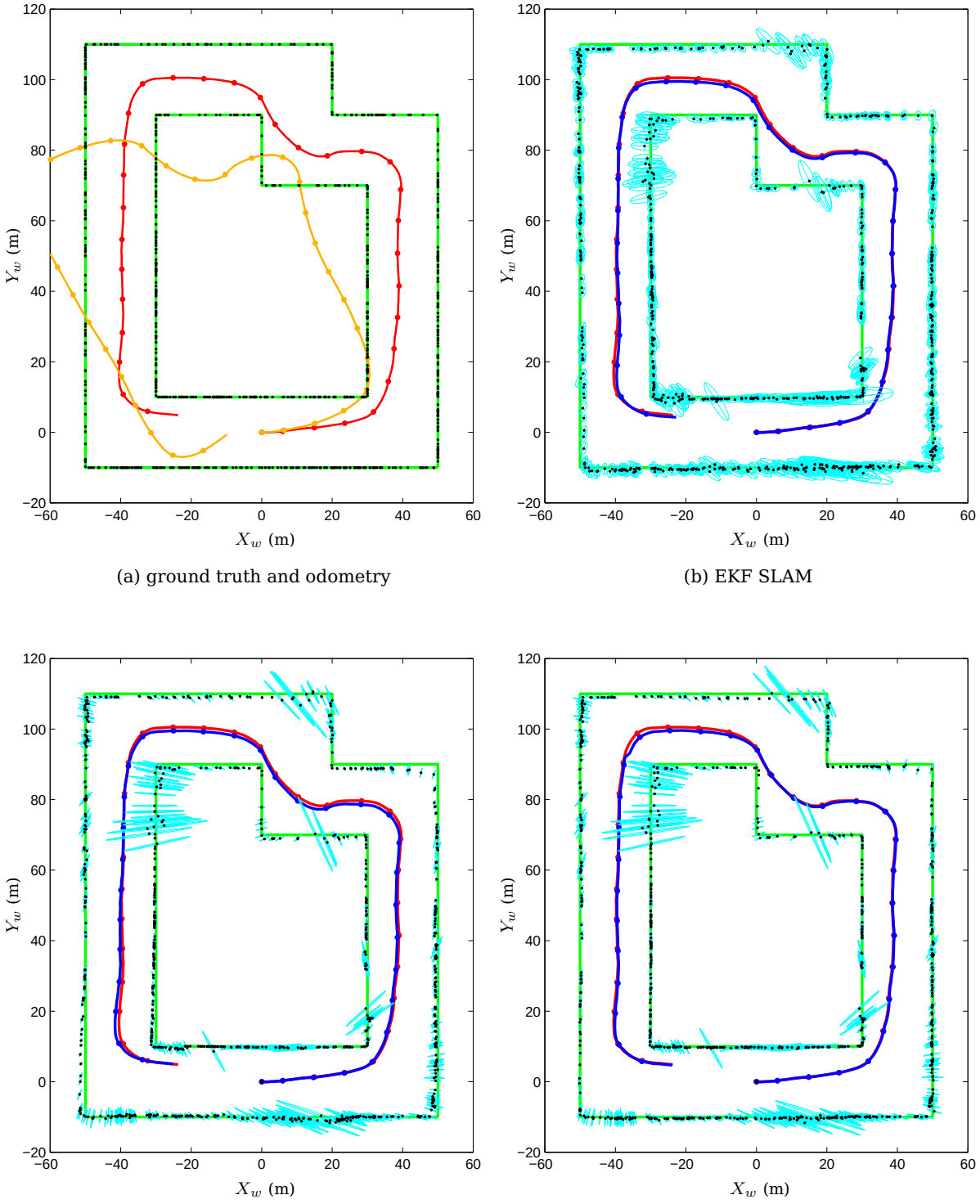



**Figure 6.1** – The Euclidean error of EKF SLAM (red), FastSLAM (blue) with 250 particles and FastSLAM 2 with 80 particles from the 2D simulation where landmarks had a 40% chance of being observed when inside the field of view.



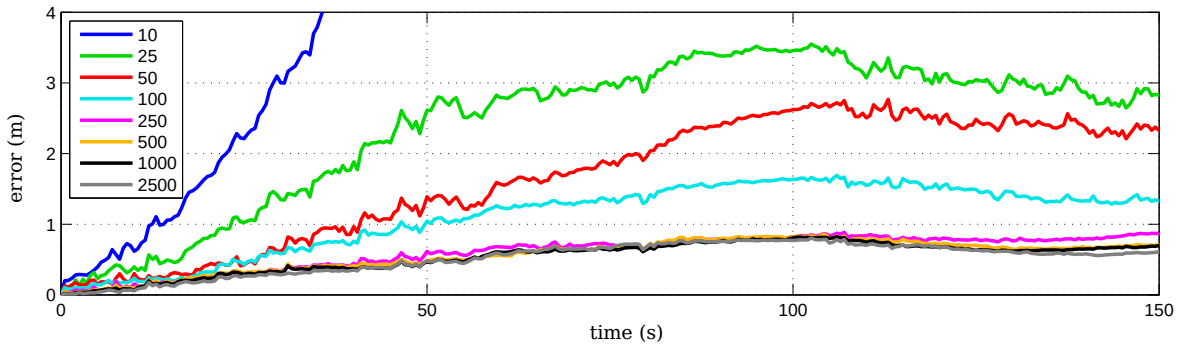

(a) ground truth and odometry

(b) EKF SLAM

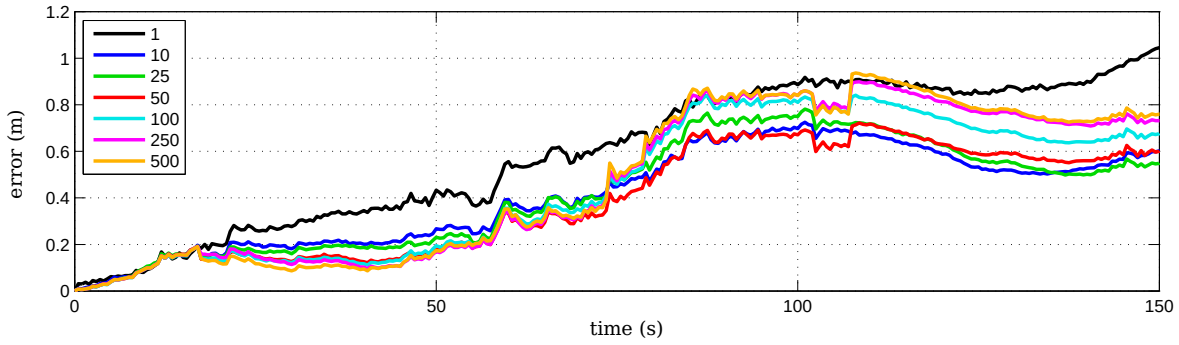
(c) FastSLAM

(d) FastSLAM 2

**Figure 6.2** – The routes and maps from the 2D simulation of the three SLAM algorithms. The ground truth route is shown in red and the environment walls in green. (a) also depicts the route calculated using only the control update in orange and the true locations of landmarks as black dots. (b), (c) and (d) depict the estimated routes in blue and the estimated landmark positions as black dots with corresponding confidence ellipses in cyan. During these simulations landmarks had a 40% chance of being observed when in the field of view. All trajectories are shown with markers on every tenth time step. The robot started at the origin.




(a) FastSLAM




(b) FastSLAM 2

**Figure 6.3** – The effect of different numbers of particles on the Euclidean error of the routes estimated by the two FastSLAM algorithms. Note the difference in scale of the error axes.

end of each test with the total distance, similar to Civera et al. [9]. This parameter provides an indication of the drift at the end of the run, but may be misleading as it holds little information about the error during the test. For this reason we also provide the average of the percentage error over the distance calculated at every time step over the run.

| Test | SLAM algorithm | Average execution time (ms) | Estimated total distance (m) | Maximum error (m) | Percentage error over distance | Average percentage error over distance |
|---|---|---|---|---|---|---|
| 1 | EKF SLAM | 163 | 291.36 | 0.95 | 0.28 | 0.52 |
| 1 | FastSLAM | 450 | 291.91 | 0.87 | 0.02 | 0.37 |
| 1 | FastSLAM 2 | 549 | 291.50 | 0.27 | 0.06 | 0.31 |
| 2 | EKF SLAM | 85 | 293.40 | 1.35 | 0.42 | 0.69 |
| 2 | FastSLAM | 235 | 294.17 | 1.48 | 0.25 | 0.84 |
| 2 | FastSLAM 2 | 279 | 294.14 | 0.22 | 0.02 | 0.37 |
| 3 | EKF SLAM | 44 | 290.41 | 3.99 | 1.22 | 1.26 |
| 3 | FastSLAM | 120 | 293.23 | 1.05 | 0.22 | 0.59 |
| 3 | FastSLAM 2 | 132 | 291.36 | 0.53 | 0.03 | 0.45 |

**Table 6.1** – Comparison of 2D SLAM algorithms.

As expected the accuracy of each system decreases as noise increases and the number of landmarks decreases. It is interesting to note that FastSLAM 2 outperforms the other two in all three tests.

### 6.2.2 Number of particles

The previous experiment shows it is possible to achieve similar results using 250 particles with FastSLAM and only 80 with FastSLAM 2. To further investigate the relationship between the number of particles and the accuracy of these two algorithms we run several tests, using the 2D environment, with different numbers of particles. For every such number we ran the test 20 times in an attempt to remove randomness introduced by the pose sampling step of the algorithms. Results of these experiments for FastSLAM and FastSLAM 2 are shown in Figure 6.3.

We see that with FastSLAM in 2D, 250 particles is a good number to use as we do not lose much accuracy when compared using to a larger number of particles. Rather surprisingly, the accuracy of FastSLAM 2 changes very little with changes in the number of particles. Even when using just one particle the algorithm consistently produces accurate results. Intuitively this may seem incorrect but Montemerlo et al. [26] show that we can expect this behaviour.

### 6.2.3 Landmark identification

There are two major concerns when working with feature detectors that we should investigate. Firstly we expect that some percentage of features will not be observed every time that they are inside the field of view and, secondly, we expect feature mismatches. We can simulate both these problems and investigate their effect on accuracy.

We show the results for EKF SLAM in Figure 6.4. We see that when we change the likelihood that a landmark would be observed, accuracy starts to decrease when that likelihood falls below 50%. It is important to note that even when the likelihood of landmarks being observed falls below 10%, the resulting accuracy is still a significant improvement over a scenario where only vehicle odometry is used for localization. Results for the other two SLAM algorithms are similar.

Figure 6.5 depicts the impact that feature mismatches can have on our SLAM algorithms. We ran a test similar to the one in Section 6.2.1, but added six feature mismatches over three time steps. The EKF displays erratic behaviour with very large errors. In a practical situation, such large errors can be catastrophic to an estimation system. Although FastSLAM shows significant drift, it remains stable. FastSLAM 2 copes remarkably well with the mismatches and introduces only a small amount of drift.

In order to test our novel probabilistic outlier removal scheme (Section 4.3), we implemented it with the EKF SLAM simulation and used it to find and remove landmark mismatches in corrupted

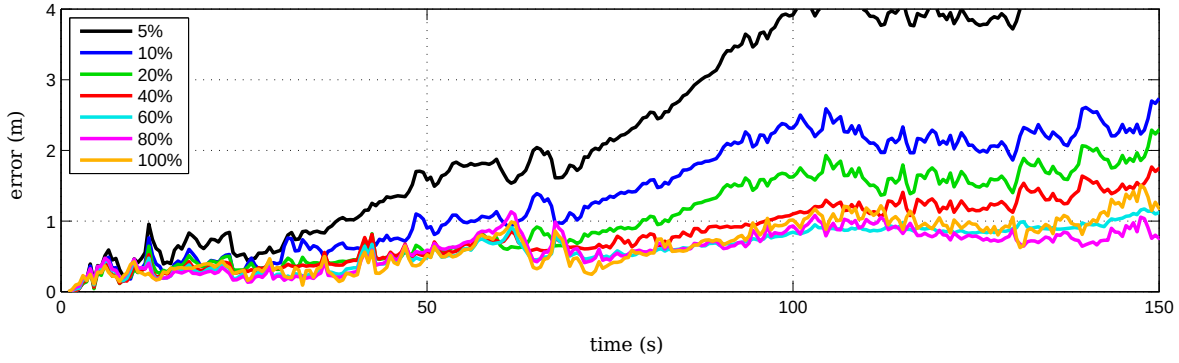
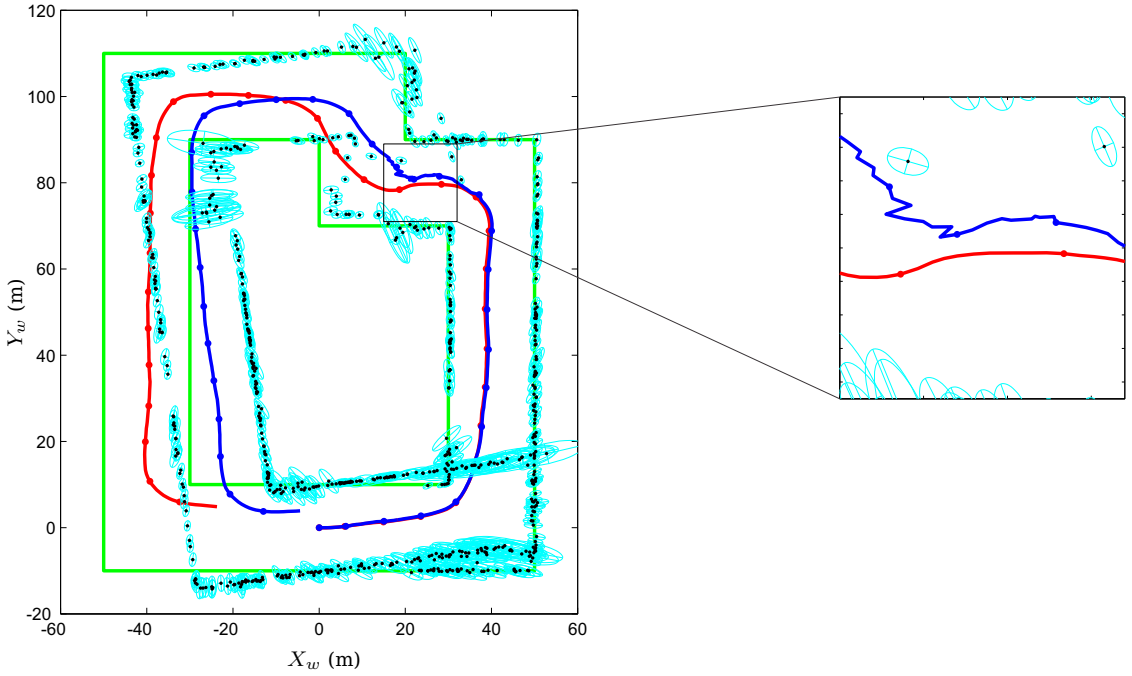
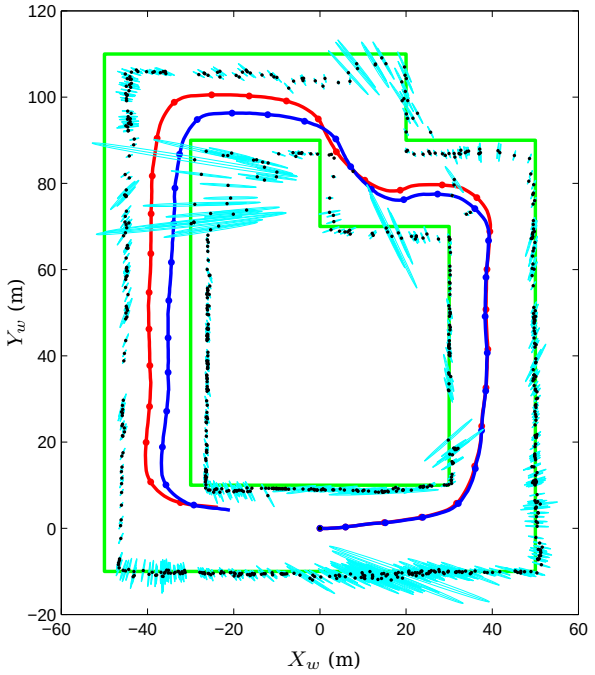


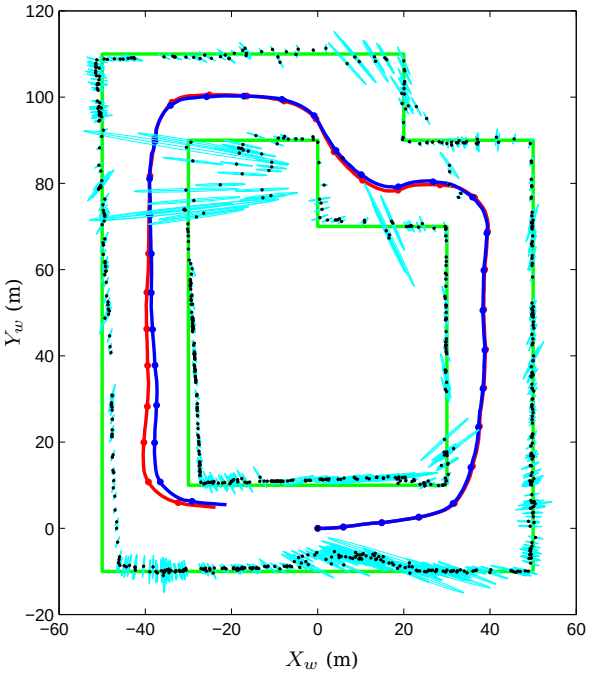

**Figure 6.4** – The effect of different probabilities that landmarks will be observed while in the field of view on the Euclidean error of the routes estimated by EKF SLAM.




(a) EKF SLAM




(c) FastSLAM




(d) FastSLAM 2

**Figure 6.5** – The effect of landmark mismatches on the route and map estimation of the three SLAM algorithms. The route estimate of EKF SLAM is shown with an enlarged section were the estimate becomes erratic. The ground truth route is once again shown in red, with the actual walls on which the landmarks should lie in green. Estimated routes are shown in blue with the estimated landmarks in black with cyan confidence ellipses.

measurements. We chose mismatches at random according to different likelihoods. For every likelihood we ran the test 30 times to remove randomness introduced by the choosing of outliers. We show the results of this experiment in Figure 6.6. Note that for up to 15% chance of a landmark measurement being corrupted the error remains low, but that it increases from there. Fortunately 15% is far more than what we expect with real measurements. Also note that despite a large number of outliers, the EKF remains stable. Any loss in accuracy here is due to the low number of landmarks than can be used, rather than mismatches.

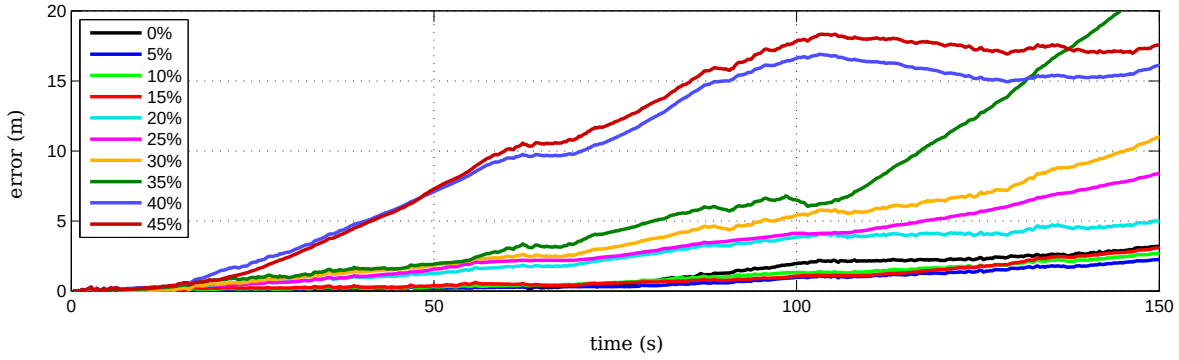



**Figure 6.6** – The effect of landmark mismatches on the Euclidean error of EKF SLAM when using our probabilistic outlier removal scheme. The percentage indicates the chance of a landmark being identified incorrectly.

We conducted another test in order to compare our probabilistic outlier detection scheme with the classic fundamental matrix-based method. As the robot moves around the simulation environment we find correspondences between every new measurement and the previous one. We then introduce 10% mismatches at random in the correspondence vector at every time step. By counting the inliers and outliers correctly identified by both outlier detection algorithms we are able to compare the two. The results of this experiment are shown in Table 6.2.

| Outlier detection algorithm | Average percentage correct inliers identified | Average percentage correct outliers identified |
|---|---|---|
| Probabilistic 3D | 100 | 99.90 |
| Fundamental matrix (strict) | 25.13 | 98.56 |
| Fundamental matrix (permissive) | 65.72 | 82.96 |

**Table 6.2** – Comparison of outlier detection schemes.

We see that in order to use the fundamental matrix-based method we have to set the threshold very strictly. This means that only a fraction of the inliers can be identified correctly. By increasing the threshold a large number of outliers are misclassified. In this test our method, on the other hand, does not misclassify any inliers and finds nearly all the outliers.

### 6.2.4 3D SLAM

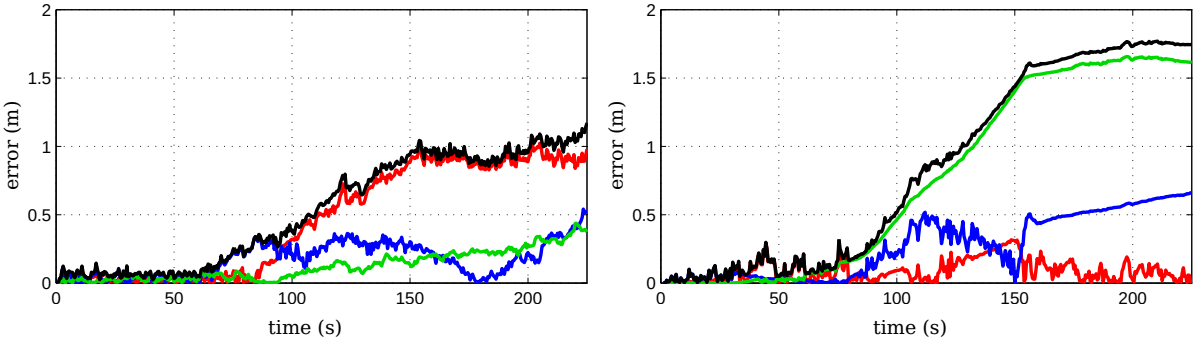
Finally we conduct a few simulations in the 3D environment. These tests are conceptually the same as the 2D tests, except that the robot has six degrees of freedom and every landmark has three.

The EKF SLAM algorithm performs similarly to the 2D case and produces highly accurate results. We again had trouble with stability and had to be very careful in choosing suitable noise parameters.

Accuracy is depicted in Figure 6.7 with the map and ground truth in Figure 6.8. We notice that the uncertainty of the location of the robot grows throughout the simulation, which in turn increases the uncertainty in landmark locations.

In the 2D case we find 250 to be a suitable number of particles to describe the three robot states with FastSLAM. To have the same accuracy with six states we theoretically need more than 60 000 particles which is impossible from a practical point of view. In fact, we encounter memory problems when working with 5 000 particles, not to mention the execution time. Through simulation we find that such a high number may indeed be required as we are unable to obtain a degree of accuracy that would be practically beneficial to a system. For this reason we do not provide any results for FastSLAM in the 3D case.

FastSLAM 2, on the other hand, performs well in 3D. We again notice that the algorithm yields accurate results with only one particle. At one point the algorithm does make a small error in the pitch angle, which introduces drift in the $z_t$ state, as we can clearly see in Figure 6.7 (b). Overall accuracy, however, remains within reasonable bounds.




(a) EKF SLAM (b) FastSLAM 2

**Figure 6.7** – Location errors for EKF SLAM and FastSLAM 2 from the 3D simulation. The error in $x_t$ is shown in red, $y_t$ in blue, $z_t$ in green and the total Euclidean error in black.

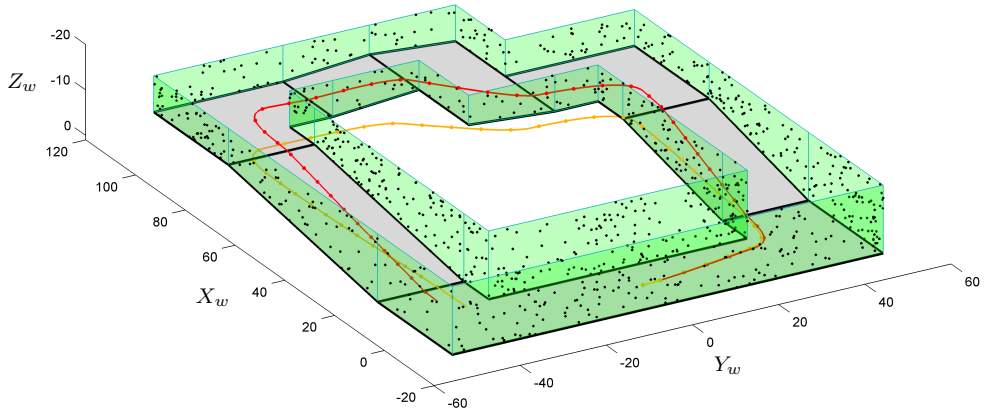
## 6.3 Discussion

The purpose of the simulations described in this chapter was to establish if the SLAM algorithms we described would be suitable for use with triangulated image features as landmarks.

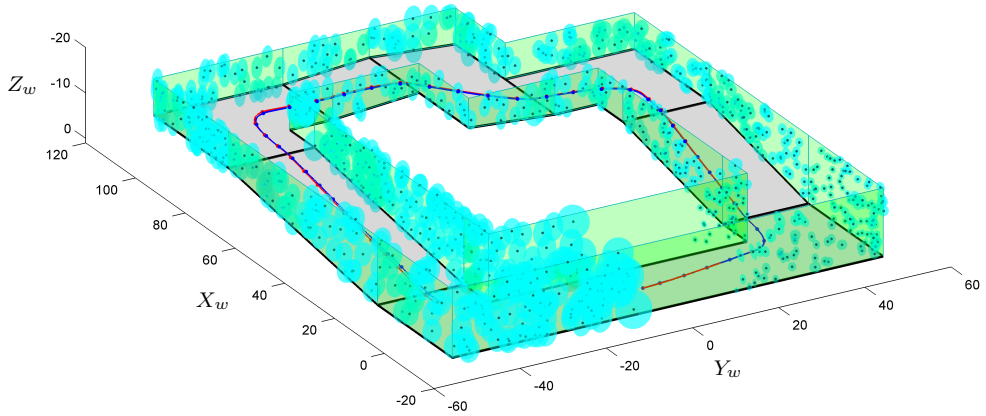
We find that in the 2D case all three SLAM algorithms can achieve accurate results, especially FastSLAM 2. In order to test a more realistic representation of feature detectors we test the accuracy when features have a chance of not being seen. Again all three algorithms perform well. When we introduce mismatches in landmarks, EKF SLAM displays extreme sensitivity. By using our probabilistic outlier detection scheme, which is remarkably adept at its task, this issue can be remedied. Compared to EKF SLAM the other two algorithms perform better in the presence of outliers.

FastSLAM displays the expected inaccuracy in 3D as it is not feasible to use the appropriate number of particles to accurately describe a six-dimensional distribution. In contrast to FastSLAM, FastSLAM 2 performs very well with very few particles.

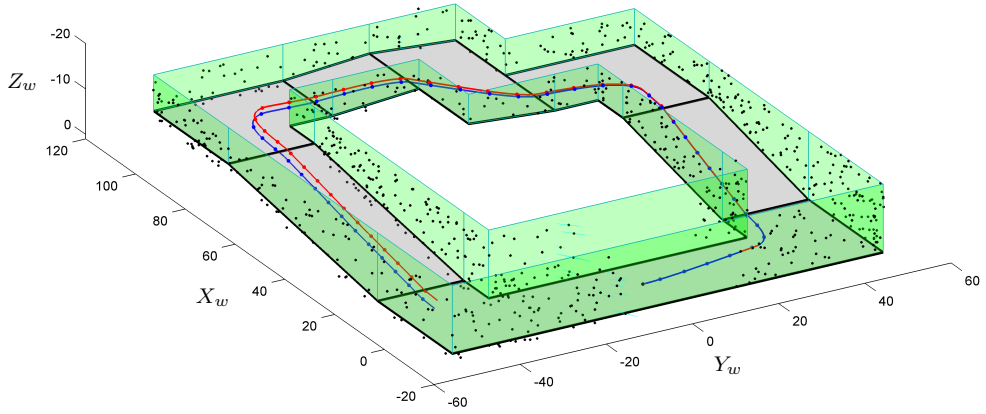
In conclusion we note that the EKF SLAM algorithm works well when correct feature matches can be ensured. The outlier removal scheme we developed proved to be able to do so to a high degree. There are, however, other problems related to the algorithm such as sensitivity to parameter tuning and it being susceptible to becoming unstable.




(a) ground truth




(b) EKF SLAM




(c) FastSLAM 2

**Figure 6.8** – The routes and maps from a 3D simulation of EKF SLAM and FastSLAM 2. The ground truth route is shown in red and the environment walls in green. (a) also depicts the route calculated when only the control update is used in orange (note that it remains on a 2D plane) and the true locations of landmarks as black dots. (b) and (c) depict the estimated routes in blue with the estimated landmark positions as black dots with corresponding confidence ellipsoids in cyan. Landmarks have a 60% chance of being observed when in the field of view.

FastSLAM is a viable option in 2D as it is easy to implement and generally highly accurate. Its inability to handle the 3D case is, however, a major drawback.

FastSLAM 2 seems to encompass the strengths of the other two algorithms in that it is as accurate as EKF SLAM, but as robust and stable as FastSLAM. Even though we did not conduct a thorough study of execution time it is safe to say that the low number of particles needed by FastSLAM 2 also enables it to execute considerably faster than the other two algorithms.

Now that we have an idea of the pitfalls that we can expect, we can proceed to testing with real data. The next chapter provides results from several datasets using the different feature detection algorithms, outlier removal schemes and SLAM algorithms.

# Chapter 7

# Experimental results

The final step in our investigation and development of a SLAM system that uses stereo vision as a sensor is to test the complete system with real world data. In this chapter we experiment with several different permutations of the system components, in order to establish how well they perform and to discover whether it is feasible to use our system in a practical sense.

The chapter starts with a description of the test platform we developed and the data we captured. We then compare SLAM in 2D with SIFT and SURF, and report some interesting observations. Finally we present 3D SLAM results and follow with an overall discussion of the results of our experiments with real world data.

## 7.1 Experimental setup

We start this section with a description of the robot we used and some of the design choices we made, and proceed with a description of the datasets we captured.

### 7.1.1 Test platform

Each real world dataset should ideally consist of a set of images captured by two synchronized and calibrated cameras, control inputs and independently obtained ground truth location information that can be used to evaluate the algorithms.

In order to capture such datasets we mounted a stereo camera set on a mobile robot. We used a Pioneer 3-AT from Mobile Robots and the Advanced Robot Interface for Applications (ARIA) C++ library, freely available online [24]. We programmed the robot to execute a command given to it by a human using a joystick controller. At every time step we store the forward and rotational velocities, calculated by ARIA using the rotational speed of the wheels, to be used as control input by the SLAM algorithms.

Our stereo camera rig consists of two Point Grey Firefly MV cameras with a synchronization unit we developed. At every time step the synchronization unit prompts the cameras to capture a pair of images. The images are then rectified (Section 3.2.3) and stored. When considering the speed of the robot, we chose a frequency of 4 Hz.

The baseline (distance) between the cameras is an important design parameter. If the baseline is too small the uncertainty in measurements becomes very large, but if it is too big the sensor will be unable to observe features close to the robot with both cameras. For our purposes we found 30 cm to be a suitable trade-off. Choosing lenses for the cameras presents a similar kind of decision. If the

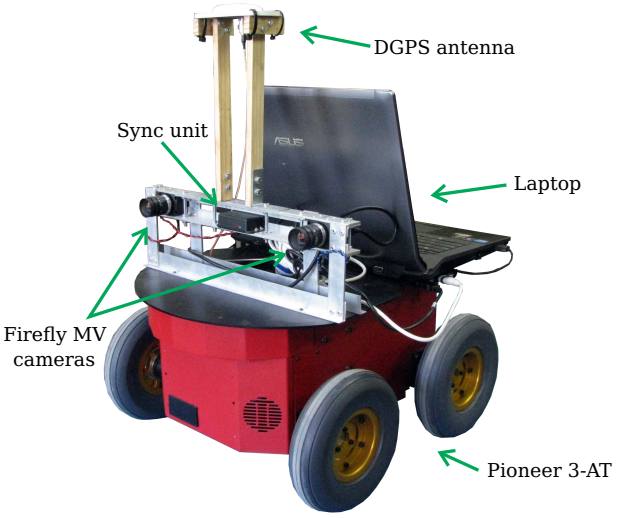



**Figure 7.1** – Test platform.

lens is too long the field of view becomes too small, but if the lens is too short measurement noise increases. We opted for 6 mm lenses.

Ground truth data was recorded with a DGPS (accurate to about 5 cm) mounted on the robot. Note that this ground truth data is not used in any of our SLAM systems, and is employed merely for comparing results afterwards.

Figure 7.1 shows a picture of the test platform with its components.

The algorithms are implemented as they had been in the simulations, except that feature detectors are included for the real world situation.

### 7.1.2 Datasets

When we work in a real world scenario we should expect problems such as bad lighting, uncluttered scenes (that give very few features), and a fair amount of shaking. We tried to capture realistic datasets that include these problems to a degree.

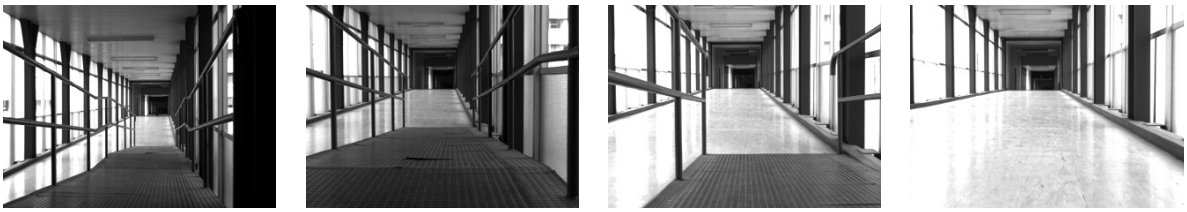
The first two datasets were captured on the roof of the Electrical and Electronic Engineering building in Stellenbosch. The roof is a suitable environment to test 2D SLAM algorithms, since it is more or less flat. Apart from trees moving in the wind it is also completely static. For both these datasets we also captured ground truth with the DGPS.

The first of the two roof datasets includes a fair amount of maneuvering around two obstacles over a distance of about 45 metres. The second dataset comprises of a slow turn, a fairly long straight section, a three point turn with some reversing, and another straight section. The robot covered about 70 metres. Note that turning increases the process noise substantially because of wheel slippage.

A third dataset was captured to test the 3D SLAM algorithms. We used a wheelchair ramp for this purpose. Since the ramp is indoors we were unable to capture DGPS ground truth, so instead we measured the ramp manually to assess the error in tests with this dataset. The ramp consists of three parts: a downward incline, a short flat part and another downward incline.

A few frames of the datasets, captured by one of the cameras, are shown in Figure 7.2.

(a) outdoor roof datasets



(b) indoor ramp dataset

**Figure 7.2** – Sample images from the two outdoor roof datasets and the indoor ramp dataset.

## 7.2 Experimental results

The datasets enable us to evaluate our different systems, first in 2D and then in 3D. It is important to note that the representation of accuracy in SLAM results can be given in many forms. Civera et al. [9], Grisetti et al. [15], Dubbelman et al. [11] and Se et al. [31], for example, all use slightly different methods to analyse their results. We believe the most informative of these is to calculate the average error over the distance covered or to simply display the error in location over time (as we did in Chapter 6).

### 7.2.1 2D SLAM

We start by comparing the accuracies of EKF SLAM with our probabilistic outlier detection scheme and the classic fundamental matrix-based method (Section 4.3). This test is performed using SURF features on the second outdoor dataset. Similar to Table 6.1 we measure accuracy as a percentage of the Euclidean error over the distance travelled and as an average of this percentage taken at each time step. Results from our probabilistic method and two cases of the fundamental matrix method are shown in Table 7.1.

From the table it is clear that our method outperforms the classic method, and for this reason we use it with all the tests that follow in this chapter.

The aim of the next test is to compare the degree of accuracy we can achieve with the three SLAM algorithms. We start by testing the accuracy of the three SLAM algorithms using SURF features with our probabilistic outlier removal scheme on the first outdoor roof dataset.

| Outlier detection algorithm | Estimated total distance (m) | Maximum error (m) | Percentage error over distance | Average percentage error over distance |
|---|---|---|---|---|
| Probabilistic 3D | 70.7 | 1.26 | 0.6 | 3.3 |
| Fundamental matrix (strict) | 71.8 | 4.34 | 6.0 | 5.9 |
| Fundamental matrix (permissive) | 76.8 | 8.17 | 7.5 | 11.6 |

**Table 7.1** – Comparison of accuracies obtained using the different outlier detection algorithms.

The estimated routes with the DGPS ground truth and the estimated maps are shown in Figure 7.4, corresponding to the Euclidean errors given in Figure 7.3. We find that the SLAM algorithms are all fairly accurate to a similar degree despite the large amount of turning motion in the dataset.

Although we have no way of measuring the accuracy of the locations of landmarks in the map, we can observe a clear structure. In all three maps there are many landmarks on the obstacles around which the robot moves.

We find that the thresholds of both the feature matching and the outlier removal algorithms have to be set fairly strictly when we work with EKF SLAM to ensure that only correct matches are used, thus preventing the erratic behaviour caused by mismatches. For this reason we observe more landmarks in the maps of the two FastSLAM algorithms.

The aim of the second test is to establish the difference in performance when we use the two different feature detectors: SIFT and SURF. The discussion in Section 4.1.4 leads us to expect that SIFT would yield more accurate results than SURF. We use the second roof dataset for this test, once again in 2D. Estimated routes and maps are depicted in Figure 7.5 for the case where SIFT features are used. Euclidean errors over time are shown in Figure 7.6 for all three SLAM algorithms with SIFT (corresponding to Figure 7.5) and with SURF.

We notice that with SIFT we obtain high accuracy for all three algorithms. With SURF both FastSLAM algorithms make a small orientation error somewhere during the three point turn. This is a problem for any localization system without an absolute location measurement, since a small orientation error will inevitably lead to large location errors.

The results of the 2D experiments are summarized in Table 7.2. When we consider these results it does appear that EKF SLAM can achieve higher accuracy than the other two algorithms.

Next we investigate the ability of the algorithms to handle incorrect measurement correspondences.

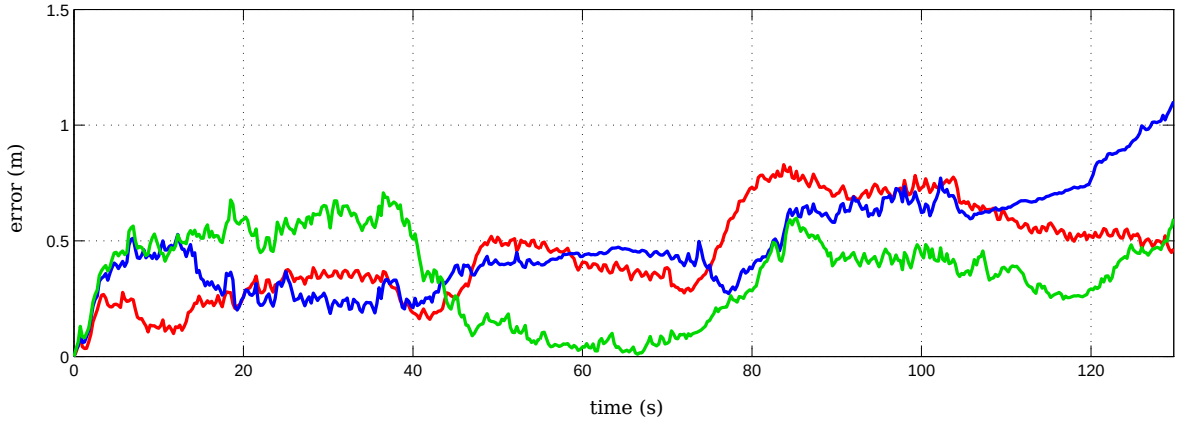
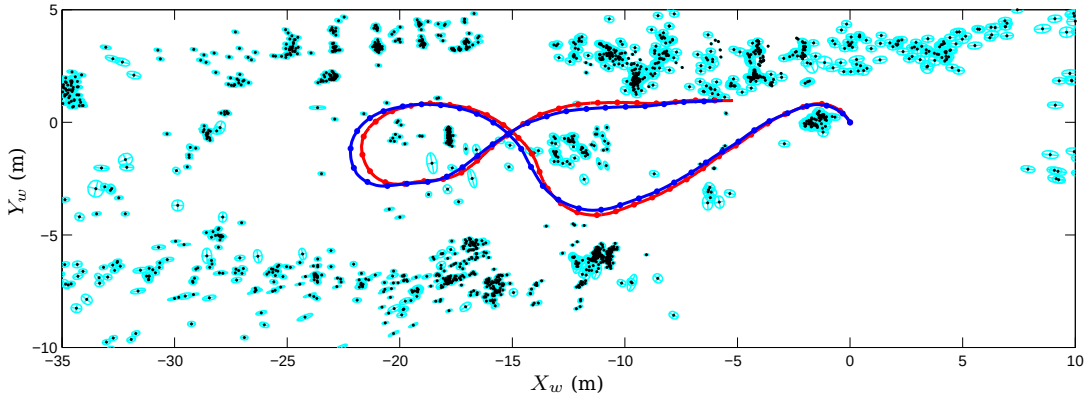
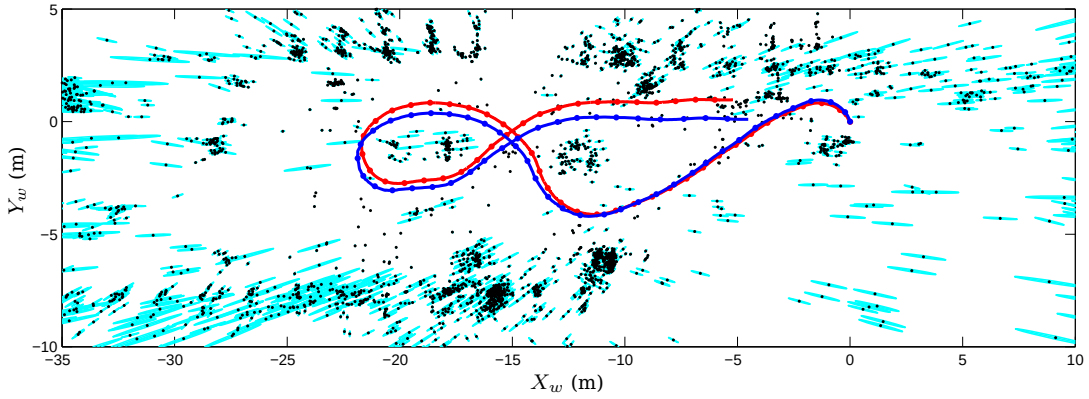


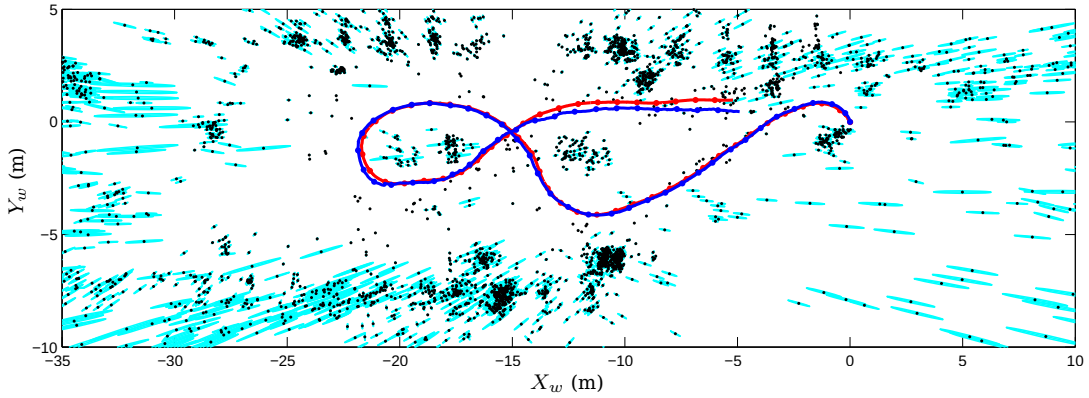

**Figure 7.3** – The Euclidean error of EKF SLAM (red), FastSLAM with 250 particles (blue) and FastSLAM 2 with 80 particles (green) using SURF features on the first roof dataset.




(a) EKF SLAM

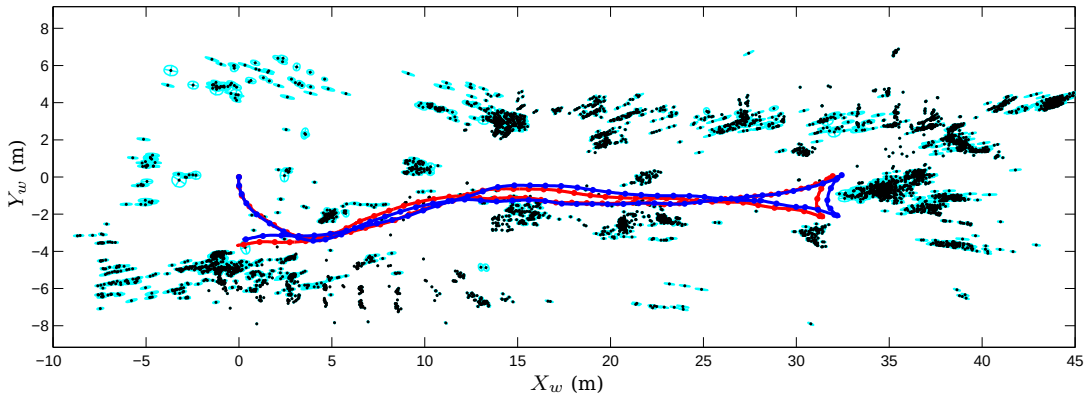



(b) FastSLAM with 250 particles

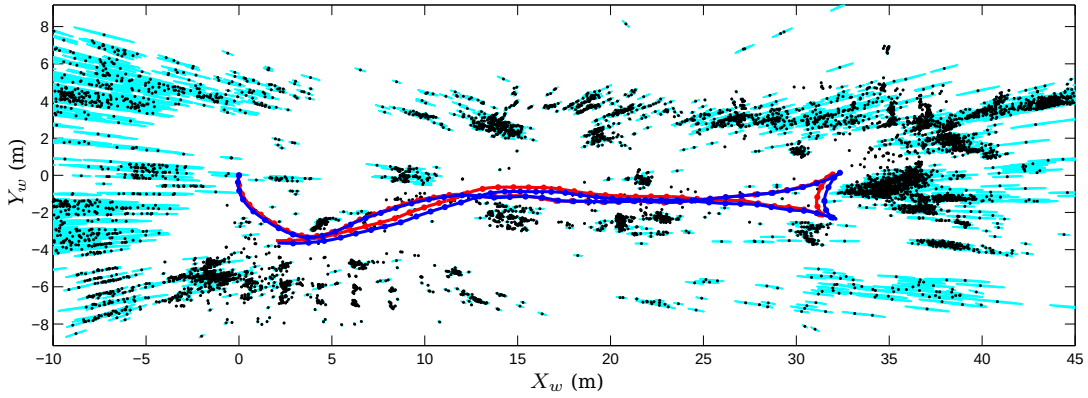



(c) FastSLAM 2 with 80 particles

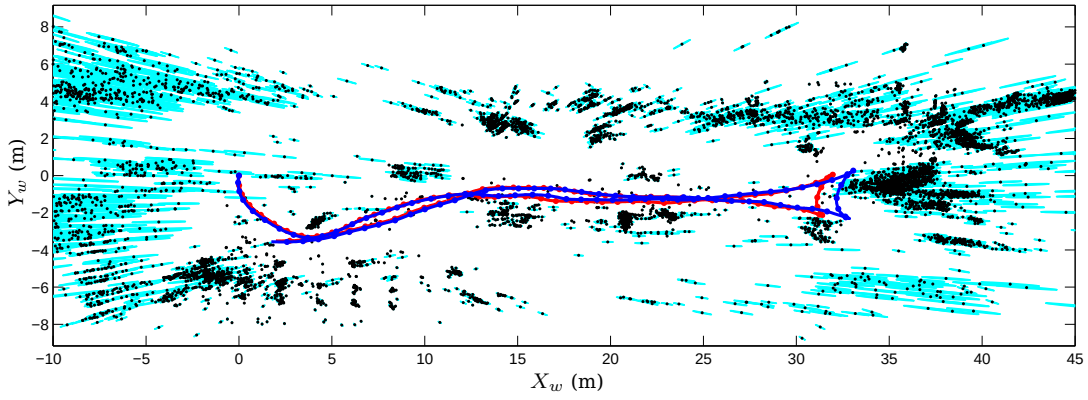
**Figure 7.4** – Estimated routes (in blue starting at the origin) and maps from the three SLAM algorithms using SURF features on the first outdoor roof dataset with the DGPS ground truth in red. Markers are placed at every tenth time step of the routes. All three algorithms use our probabilistic outlier removal scheme. The landmarks that we show as black dots with cyan confidence ellipses are those that were observed on multiple time steps, i.e. those that contributed to the accuracy of the route estimation.




(a) EKF SLAM

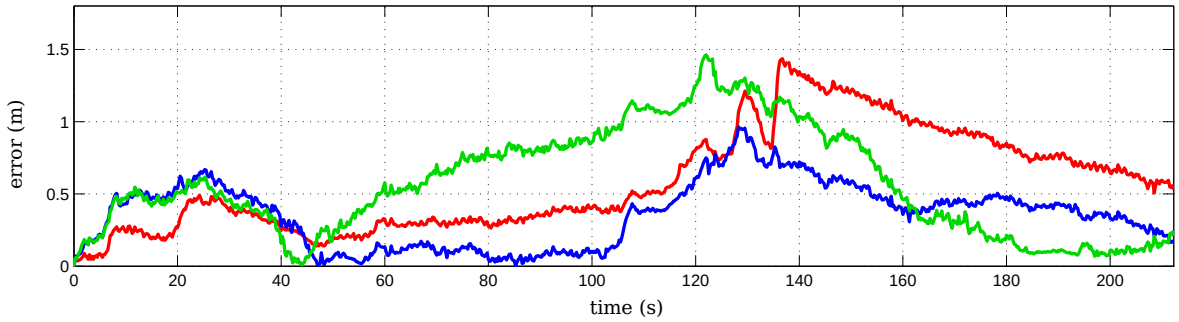



(b) FastSLAM with 250 particles

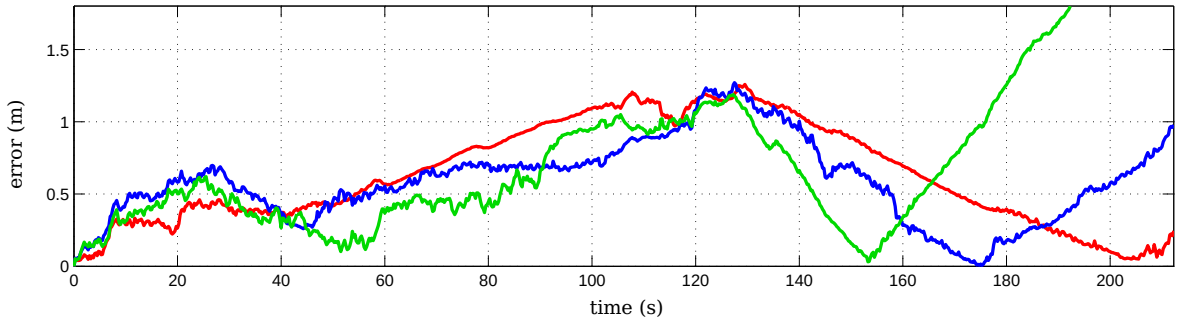



(c) FastSLAM 2 with 80 particles

**Figure 7.5** – Estimated routes (in blue starting at the origin) and maps from the three SLAM algorithms using SIFT features on the second outdoor roof dataset with the DGPS ground truth in red. Markers are placed at every tenth time step of the routes. All three algorithms use our probabilistic outlier removal scheme. The landmarks that we show as black dots with cyan confidence ellipses are those that were observed on multiple time steps, i.e. those that contributed to the accuracy of the route estimation.




(a) SIFT




(b) SURF

**Figure 7.6** – The Euclidean error of EKF SLAM (red), FastSLAM with 250 particles (blue) and FastSLAM 2 with 80 particles (green) using different feature detectors on the second roof dataset.

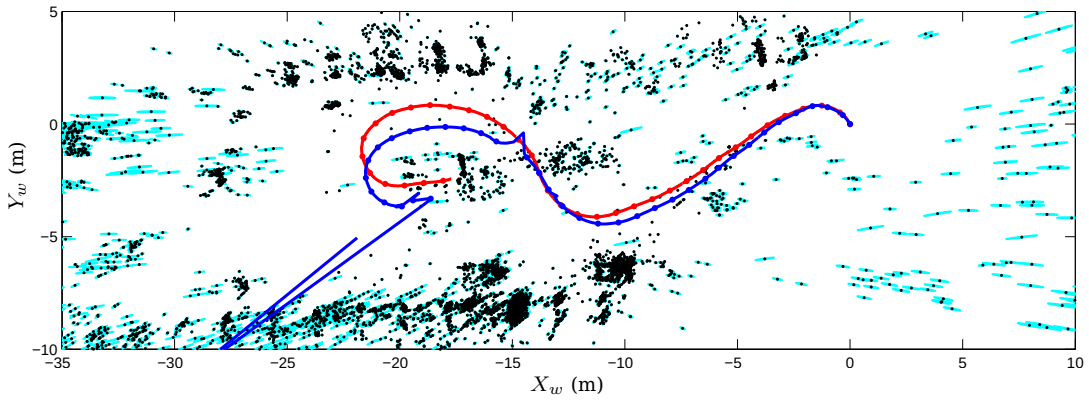
| Dataset | SLAM algorithm | Feature detector | Estimated total distance (m) | Maximum error (m) | Percentage error over distance | Average percentage error over distance |
|---|---|---|---|---|---|---|
| 1 | EKF SLAM | SURF | 44.9 | 0.83 | 1.0 | 3.6 |
| 1 | FastSLAM | SURF | 45.6 | 1.10 | 2.4 | 4.8 |
| 1 | FastSLAM 2 | SURF | 45.9 | 0.71 | 1.3 | 5.4 |
| 2 | EKF SLAM | SIFT | 70.5 | 1.44 | 0.5 | 2.7 |
| 2 | FastSLAM | SIFT | 68.6 | 0.96 | 0.2 | 3.6 |
| 2 | FastSLAM 2 | SIFT | 70.3 | 1.46 | 0.3 | 4.3 |
| 2 | EKF SLAM | SURF | 70.7 | 1.26 | 0.6 | 3.3 |
| 2 | FastSLAM | SURF | 70.0 | 1.27 | 1.4 | 4.1 |
| 2 | FastSLAM 2 | SURF | 69.2 | 3.09 | 4.5 | 4.0 |

**Table 7.2** – Summary of 2D SLAM results.

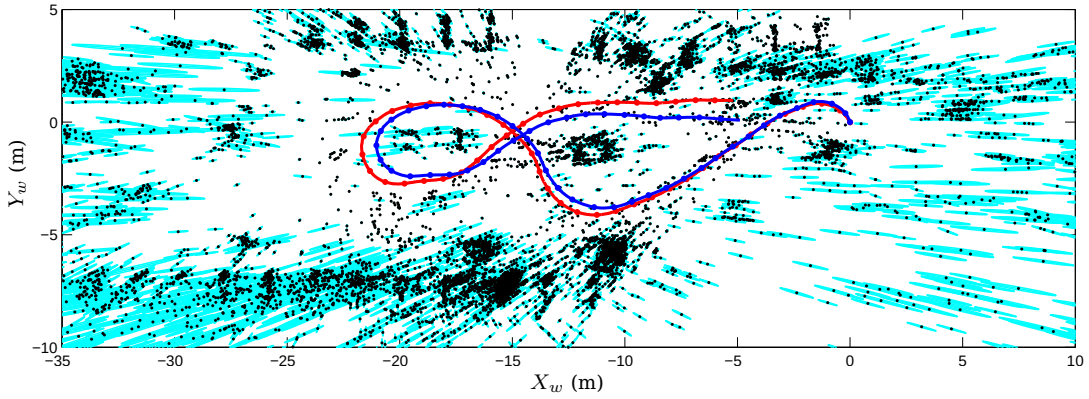
Although we take several precautions to ensure correct measurement correspondence, it may still be possible for mismatches to occur. By choosing the matching and outlier removal thresholds poorly we increase the likelihood of mismatches. The resulting route and map estimates for EKF SLAM and FastSLAM are depicted in Figure 7.7.

The effect of the outliers is as expected and similar to the effect we observed in simulation. Initially EKF SLAM makes some mistakes, but seems able to recover to some extent. Eventually, though, it becomes completely unstable. With exactly the same measurements FastSLAM remains stable and fairly accurate, except for a degree of drift. In this test FastSLAM 2 produces results very similar to those of FastSLAM.

During the tests we noticed an interesting consequence of using particles. We mentioned that particles can describe any distribution, and we observe something to this effect in Figure 7.8. The




(a) EKF




(b) FastSLAM with 250 particles

**Figure 7.7** – Poorly chosen matching and outlier removal thresholds are used to investigate the effect of mismatches.

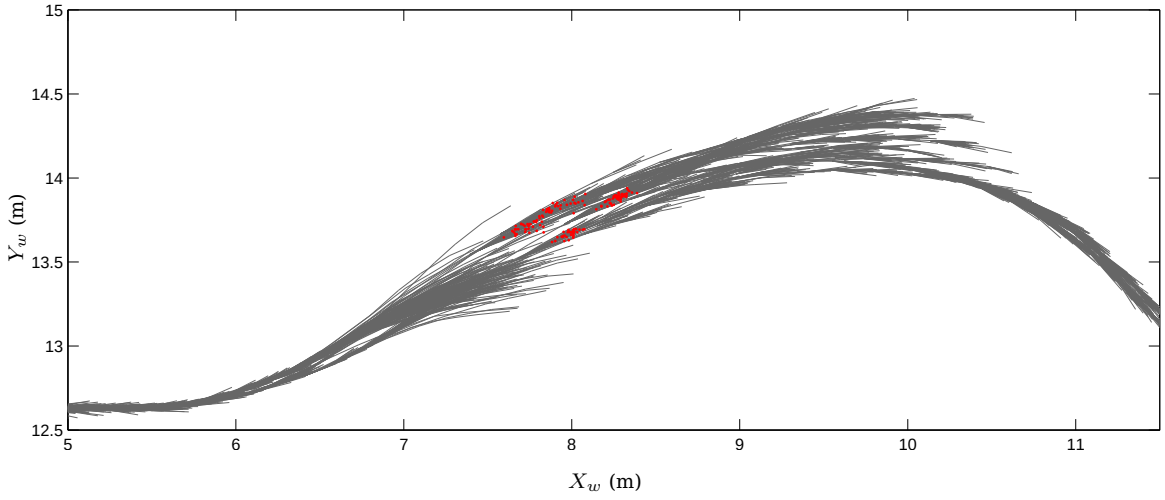



**Figure 7.8** – FastSLAM multi-modal behaviour. The robot moves from the left side of the figure to the right. Particles at one time step are highlighted as red markers, with the routes of the different particles over the time interval in grey.

robot passes through a dark area with very few measurements. As expected the uncertainty grows as the particles begin to spread out, and we can clearly see a multi-modal distribution with three lobes. As the robot progresses some of the lobes die until only one survives.

### 7.2.2 3D SLAM

The final test is set up to evaluate the performance of our systems in 3D. We use the wheelchair ramp dataset for this test. The resulting routes and maps of EKF SLAM and FastSLAM 2 are depicted in Figure 7.9, as well as the height $z_t$ and pitch angle $\theta_t$ robot states.

Once again we obtain results very similar to those from our simulations. Both algorithms are successful in estimating the change in pitch angle caused by the ramp. It is important to note that when the robot moves forward the motion is very noisy in the pitch angle, but that the drift remains fairly small over the ramp in spite of the noise (generally to within one degree of what we expect). Because we measured the ramp manually we are unable to account for the uneven surface, and the results should be viewed as such. The high peak in pitch angle at the end of the second incline shown in Figure 7.9 (d), for instance, is caused by the uneven transition between the ramp and the ground surface, and not because of an error.

## 7.3 Discussion

A first conclusion that we can draw from the results presented in this chapter is simply that all three algorithms, whether with SIFT or SURF features, are able to produce fairly accurate results. Although some drift is present, the accuracy is comparable to that of other state-of-the-art systems [9] [11].

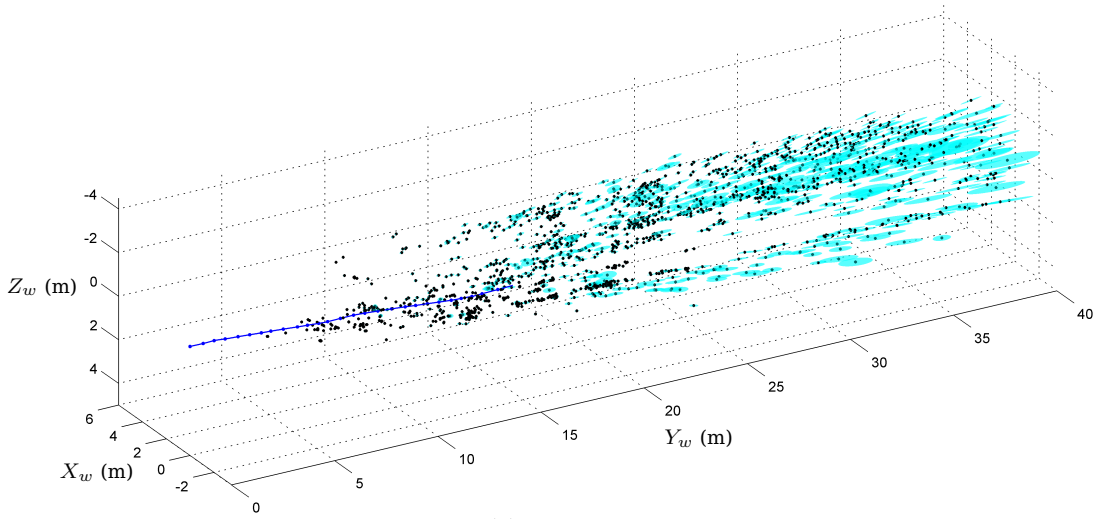
Regarding the choice between SIFT and SURF, we conclude that both can produce accurate results although SIFT is slightly more accurate at the cost of longer execution time.

It is clear that our probabilistic outlier detection algorithm outperforms the fundamental matrix-based method. We also note the importance of having an effective outlier detection scheme for the accuracy of a SLAM system that uses image features as landmarks.

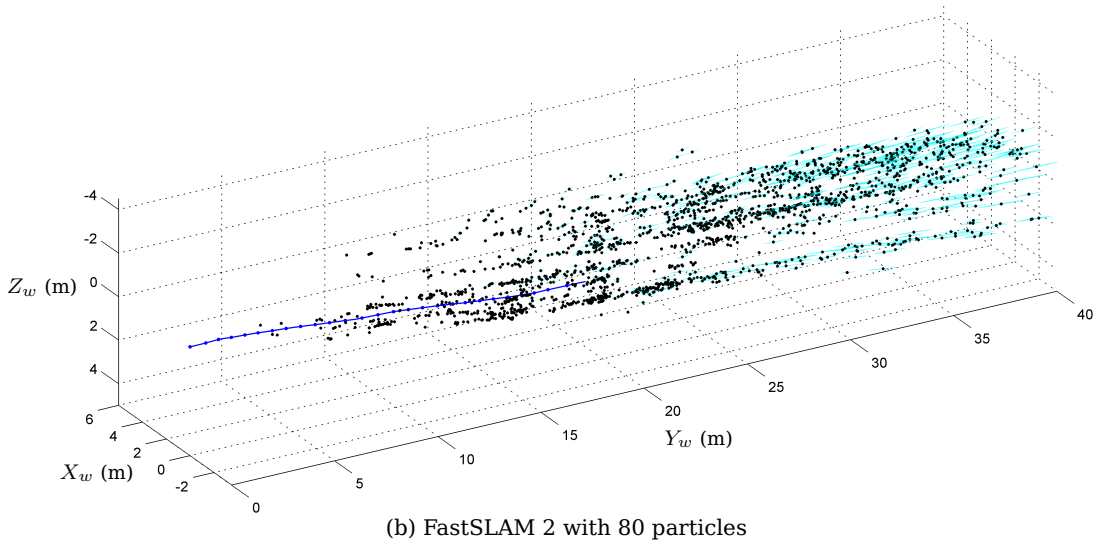
The three SLAM algorithms achieve remarkably similar accuracies in our tests. Although EKF SLAM can be slightly more accurate than the FastSLAM algorithms, the fact that it is unable to handle landmark mismatches remains a distinct disadvantage.

When choosing between FastSLAM and FastSLAM 2, the application must be considered. FastSLAM is easy to implement and typically very accurate and stable. The number of particles needed may, however, be a disadvantage. If the process noise is high, or if we are working in 3D, the number of particles needed can quickly become too high. This is where FastSLAM 2 holds the advantage since only a few particles are needed.

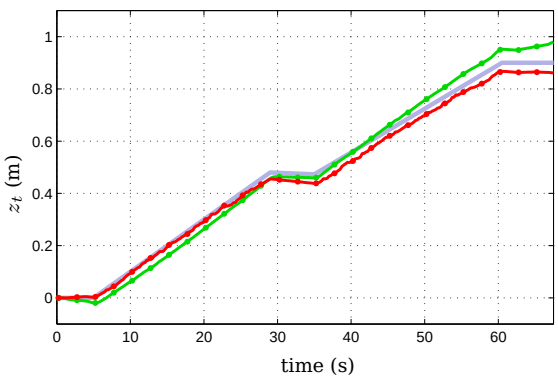
Finally it should be noted that the results obtained in this chapter are in every instance almost exactly what the simulations had led us to expect, which implies that our simulation results can be trusted and that conclusions drawn from them are valid.



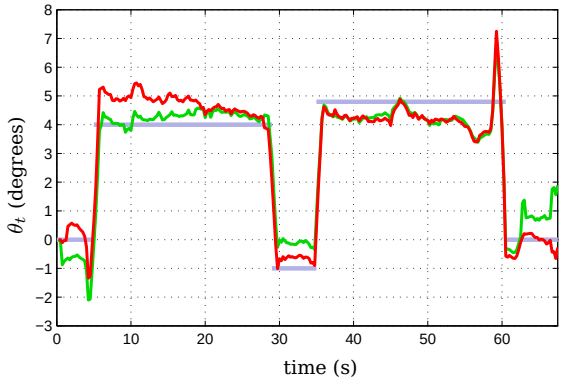

(a) EKF SLAM




(b) FastSLAM 2 with 80 particles




(c) $z_t$ state




(d) $\theta_t$ state

**Figure 7.9** – Results from the 3D SLAM systems with the ramp dataset. The estimated routes are shown as blue lines with markers on every tenth time step. Landmarks are depicted as black dots with cyan confidence ellipsoids. The $z_t$ and $\theta_t$ states are depicted in (c) and (d), for EKF SLAM in red and FastSLAM 2 in green. Manually measured height and pitch angles are depicted as grey lines.

# Chapter 8

# Conclusions and future work

The aim of this thesis was to explore the use of stereo vision as a sensor for SLAM. We designed a complete SLAM system and put it to the test in simulation and with real world data. With this final chapter we take a step back in order to draw some conclusions and present some ideas for further work.

## 8.1 Summary and conclusions

We started with a literature study. By considering several publications from the field we were able to establish a structure that most stereo vision-based SLAM systems follow: features are extracted from stereo images and triangulated to 3D landmarks, these landmarks are matched with ones stored in the map, mismatches are detected, and SLAM is performed. In order to design our system we decided to investigate each of these steps in detail.

The first step was to describe the geometry of a stereo vision sensor. Although two-view geometry is regarded to be solved on a theoretical level, and we employ the Camera Calibration Toolbox for MATLAB [5] to perform our calibration, it is important to understand the basic principles. We described the standard approach of using the pinhole camera model (with a Taylor approximation to model radial distortion) and performing basic calibration. By rectifying incoming stereo image pairs we were able to simplify the process of converting a pair of image coordinates to a 3D point. Rectification and the epipolar constraint also mean that a 3D point captured by the two cameras have the same vertical coordinates in both images.

By linearizing the transformation between stereo image features and 3D landmarks, we were able to derive a Gaussian measurement noise model. We tested the model by comparing it to real world noise and found that it produced similar distributions.

In order to find landmarks in images we opted for a feature detection algorithm. The first task of such an algorithm is to find salient points in images that will hopefully be detected despite changes in the point of view or illumination. The second task is to calculate a descriptor for each feature that is invariant to scale, in-plane rotation and illumination. We studied two algorithms to perform these tasks: the scale invariant feature transform (SIFT) and speeded-up robust features (SURF).

Both feature detection algorithms perform well with the SLAM systems. The main difference between the two is that SIFT is slow to execute but generally very accurate while SURF is fast but slightly less accurate. This result is exactly what we expected, and the choice between the two is made easy: for real-time applications we recommend SURF, and for off-line systems SIFT.

We found that accurate landmark matching is very important in the SLAM systems. We developed an extensive matching procedure using feature descriptors and a number of constraints. This procedure is able to correctly match most features, but makes a few mistakes from time to time. In order to identify these feature mismatches we developed a novel probabilistic consensus measure, using our measurement noise model, for use in a RANSAC framework. This outlier detection algorithm performs very well, compared to the standard fundamental matrix approach, and we showed that it greatly improves the accuracy of our SLAM systems.

Once we were able to extract landmarks from images and match them over time, we shifted focus towards the SLAM problem. We described three popular algorithms in detail: EKF SLAM, FastSLAM and FastSLAM 2. The first algorithm, EKF SLAM, describes the system with one state vector containing robot pose parameters and landmark locations. By using the control input with a motion model, an EKF predicts the current pose of the robot. Then, by using the measured landmarks sequentially, the filter refines the pose of the robot and the locations of the landmarks in the map.

Through simulation and practical tests we found that EKF SLAM can be very accurate under the right circumstances. The algorithm is also relatively easy to understand and implement. These advantages are, however, outweighed by the disadvantages of high computational complexity and extreme sensitivity to landmark mismatches.

The high complexity, brought on by matrix operations on the large covariance matrix, limits the size of the map. The problem can be overcome (to a degree) by discarding old landmarks but, because measurements from a stereo vision sensor typically contain a large number of landmarks, the system can become slow.

We tested the performance of the different SLAM algorithms in the presence of landmark mismatches. EKF SLAM displayed extreme sensitivity in both simulation and practical tests. Even with our outlier detection algorithm, this problem makes EKF SLAM dangerous to use as mismatches can still occur despite our best efforts.

The second SLAM algorithm, FastSLAM, is based on the Rao-Blackwellized particle filter which uses particles to describe some states and Gaussian distributions to describe others. In the case of SLAM, the robot states are represented by particles and a Gaussian distribution is used for every landmark. The particles are updated according to the normal particle filter process, and each Gaussian distribution is updated with an EKF. This may seem excessive but is actually easy to implement and remarkable stable.

With a sufficient number of particles, FastSLAM can achieve accuracies similar to EKF SLAM. Our tests showed that 250 particles offer a good compromise between accuracy and speed in the 2D case. However, the algorithm is unable to describe the six-dimensional distribution necessary for 3D SLAM, since too many particles would be required.

FastSLAM does have the crucial benefit of being robust against landmark mismatches. The simulations and real world tests confirmed that despite a degree of drift, FastSLAM remains fairly accurate and stable in the presence of outliers. It is a significant advantage for a practical system when the algorithm can be trusted to remain stable.

The final SLAM algorithm, FastSLAM 2, is an extension of FastSLAM. By sampling particles from a proposal distribution that is created using the control input and refined using the measurements, similar to an EKF, the filter requires very few particles for accurate estimation. Tests showed that even using just one particle can be sufficient in some situations.

FastSLAM 2 has the same benefits as FastSLAM. Its accuracy is generally more or less the same as that of the other two algorithms. The small number of particles needed means that it can be

used with 3D systems, and we obtained results comparable to EKF SLAM in 3D. The performance of FastSLAM 2 is similar to that of FastSLAM when feature mismatches are introduced. Coupled with a low memory use and the potential of fast execution time (because of the small number of particles needed) this robustness to landmark mismatches makes FastSLAM 2 a very powerful and practically useful SLAM algorithm.

Our final conclusion is that we are convinced that stereo vision can be used effectively as a sensor for SLAM and our results validate further research and development.

## 8.2 Contributions

This section lists some contributions that resulted from our work. Our research contributions can be summarized as follows.

- Through tests we established that the method we use to describe uncertainty in stereo feature measurements [7] is fairly accurate. Since we use a Gaussian approximation, this model can be very useful in many probabilistic estimation problems where stereo vision is used.
- Our results offer a fair comparison between the feature detectors SIFT and SURF and the SLAM algorithms EKF SLAM, FastSLAM and FastSLAM 2.
- We showed that our novel method for finding landmark mismatches [8] is superior to the standard method of using the fundamental matrix. Our method can be very useful in other applications and is not restricted to the SLAM systems described in this thesis.
- We made several adaptations to the SLAM algorithms as they are presented by Thrun et al. [34]. We use Cartesian coordinates with our measurements where Thrun et al. use polar coordinates. We handle measurement noise in a slightly different way in that we allow each measurement to have a different covariance matrix. Our motion model is also somewhat simpler, and we derived the 3D case independently.

We also made a number of contributions that can aid further development and practical implementation.

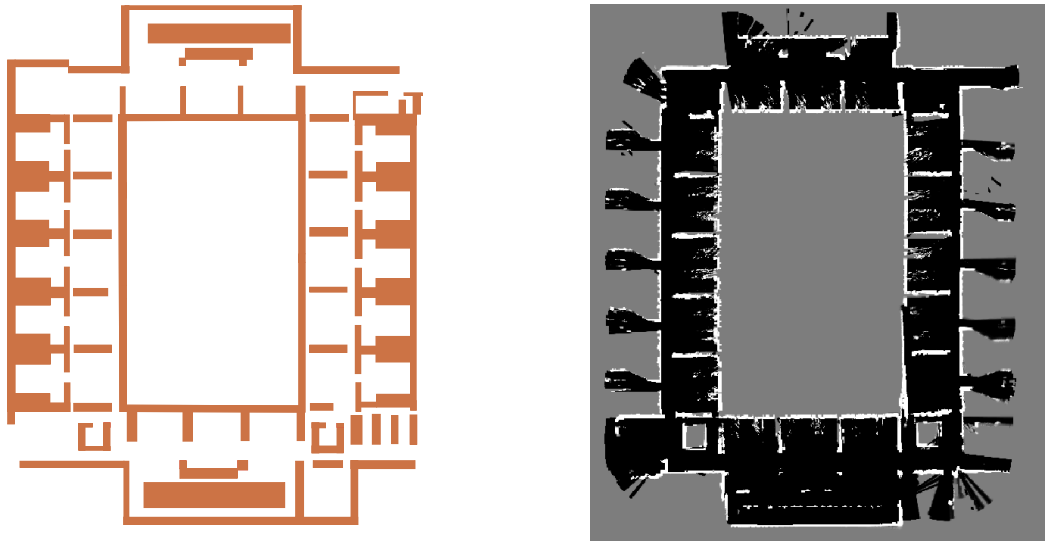
- By comparing practical tests with our simulation results, we observed that our simulation environment can be trusted as a reasonable representation of the real world. This makes it easy to test any variations we may want to apply to the system in a controlled simulation and, in doing so, eases the continuation of our research.
- The datasets we captured are also useful for future work, especially those with ground truth location data.
- Hardware development can be very frustrating and time consuming. The test platform we put together can easily be extended to include other sensors or be used in other environments to capture stereo images.

The SLAM system described in this thesis can be developed much further and can provide a starting point for those who wish to design a practical SLAM system with stereo vision.

## 8.3 Future work

There are many areas in which further research can be performed. We list a few in this section.

We mentioned in the introduction that a map of the environment can be useful for other autonomous navigation systems such as path planning. However, the maps we produce consist of point landmarks and will not be very useful for this purpose. By using other sensors with our SLAM system a more complete map can be obtained. As an example, Figure 8.1 shows an occupancy grid map built by using one of our SLAM systems and a laser range finder [18]. An occupancy grid map represents the world by discretizing it into squares for 2D or cubes for 3D. Each of these cells is then allocated a probability that it is occupied by some obstacle.



(a) schematic representation of the environment (b) estimated occupancy grid map

**Figure 8.1** – An example of an occupancy grid map created using stereo vision-based SLAM and a laser range finder. In (b) black indicates open space, white occupied.

An alternative to using an additional sensor may be dense stereo reconstruction. When a match can be found for every pixel in a stereo image pair, it may be possible to build dense maps similar to the grid map we show. Figure 8.2 depicts an example of two dense disparity maps obtained from

(a) input left image (b) graph cut disparity map (c) HDP disparity map

**Figure 8.2** – Examples of dense stereo reconstruction. Dark areas indicate a small disparity, and therefore greater depth, while light areas are closer to the cameras.

using the graph cuts algorithm [19] [40] and hierarchical dynamic programming (HDP) [32] [38]. A disparity map is a 2D representation of a 3D scene and uses the disparity $(x_L - x_R)$ between two stereo images to indicate depth (Equation 3.2.14 clarifies this concept).

Another area where we believe we can still improve is feature matching. By using a feature tracking algorithm (that encompasses some sort of optimal estimation filter) the accuracy of this process may be improved substantially.

We also believe that by using multiple hypothesis landmark tracking with FastSLAM 2 we may achieve greater accuracy. What multiple hypothesis tracking implies is that the matching of landmarks is done separately for each particle. The maps of the different particles can then be different, which increases the likelihood that some particles will make fewer or no mistakes in matching.

Because there are so many possible adaptations and possibilities that have not been explored yet, the idea of performing SLAM with stereo vision makes for an interesting problem with a promising future.

# Bibliography

[1] M. Agrawal, K. Konolige, *Real-time localization in outdoor environments using stereo vision and inexpensive GPS*, International Conference on Pattern Recogntion, pp. 1063–1068, 2006.

[2] S. Ahn, M. Choi, J. Choi, W. Chung, *Data association using visual object recognition for EKF SLAM in home environment*, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2588–2594, 2006.

[3] H. Bay, A. Ess, T. Tuytelaars, L. van Gool, *Speeded-up robust features (SURF)*, Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346–359, 2008.

[4] C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[5] J. Bougeut, *Camera calibration toolbox for MATLAB*,
available at: `http://www.vision.caltech.edu/bouguetj/calib_doc`

[6] G. Bradski, *OpenCV: open source computer vision*,
available at: `http://opencv.willowgarage.com/wiki`

[7] W. Brink, C. van Daalen, W. Brink, *Stereo vision as a sensor for EKF SLAM*, Annual Symposium of the Pattern Recognition Association of South Africa, pp. 19–24, 2011.

[8] W. Brink, C. van Daalen, W. Brink, *Probabilistic outlier removal for robust landmark identification stereo vision based SLAM*, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2822–2827, 2012.

[9] J. Civera, O. Grasa, A. Davison, J. Montiel, *1-point RANSAC for EKF-based structure from motion*, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3498–3504, 2009.

[10] A. Doucet, J. de Freitas, K. Murphy, S. Russel, *Rao-Blackwellized particle filtering for dynamic Bayesian networks*, Conference on Uncertainty in Artificial Intelligence, pp. 176–183, 2000.

[11] G. Dubbelman, W. van der Mark, F. Groen, *Accurate and robust ego-motion estimation using expectation maximization*, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3914–3920, 2008.

[12] H. Durrant-Whyte, T. Bailey, *Simultaneous localization and mapping (SLAM): part I*, IEEE Robotics and Automation Magazine, vol. 13, no. 2, pp. 99–110, 2006.

[13] M. Fischler, R. Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM, vol. 24, no. 6, pp. 381–395, 1981.

[14] G. Grisetti, R. Kuemerle, C. Stachniss, W. Burgard, *A tutorial on graph-based SLAM*, IEEE Transactions on Intelligent Transportation Systems, vol. 2, no. 4, pp. 31–43, 2010.

[15] G. Grisetti, C. Stachniss, W. Burgard, *Improved techniques for grid mapping with Rao-Blackwellized particle filters*, IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34–46, 2007.

[16] C. Harris, M. Stephens, *A combined corner and edge detector*, Alvey Vision Conference on Uncertainty in Artificial Intelligence, pp. 147–152, 1988.

[17] R. Hartley, A. Zisserman, *Multiple View Geometry*, 2nd edition, Cambridge University Press, 2003.

[18] D. Joubert, *Adaptive occupancy grid mapping with measurement and pose uncertainty*, MSc thesis, Stellenbosch University, 2012.

[19] V. Kolmogorov, Z. Ramin, *Multi-camera scene reconstruction via graph cuts*, European Conference on Computer Vision, pp. 82–96, 2002.

[20] J. Leonard, H. Durrant-Whyte, *Mobile robot localization by tracking geometric beacons*, IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 376–382, 1991.

[21] D. Lowe, *Object recognition from local scale invariant features*, IEEE International Conference on Computer Vision, pp. 1150–1157, 1999.

[22] D. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110, 2004.

[23] D. Lowe, *Demo software: SIFT keypoint detector*,
available at: `http://www.cs.ubc.ca/~lowe/keypoints`

[24] Mobile Robots, *ARIA: advanced robot interface for applications*,
available at: `http://robots.mobilerobots.com/wiki/ARIA`

[25] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *FastSLAM: a factored solution to the simultaneous localization and mapping problem*, AAAI National Conference on Artificial Intelligence, pp. 593–598, 2002.

[26] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *FastSLAM 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges*, International Joint Conference on Artificial Intelligence, pp. 1151–1156, 2003.

[27] D. Nister, O. Naroditsky, J. Bergen, *Visual odometry*, IEEE Conference on Computer Vision and Pattern Recognition, pp. 652–659, 2004.

[28] L. Paz, P. Pinié, J. Tardós, J. Neira, *Large-scale 6-DOF SLAM with stereo-in-hand*, IEEE Transactions on Robotics, vol. 24, no. 5, pp. 946–957, 2008.

[29] P. Peebles, *Probability Theory, Random Variables and Random Signal Principles*, 4th edition, McGraw-Hill, 2001.

[30] E. Rosten, T. Drummond, *Machine learning for high-speed corner detection*, European Conference on Computer Vision, pp. 430–443, 2006.

[31] S. Se, D. Lowe, J. Little, *Vision based global localization and mapping for mobile robots*, IEEE Transactions on Robotics, vol. 21, no. 3, pp. 364–375, 2005.

[32] F. Singels, *Real-time stereo reconstruction using hierarchical dynamic programming and LULU filtering*, MSc thesis, Stellenbosch University, 2010.

[33] R. Smith, M. Self, P. Cheeseman, *Estimating uncertain spatial relationships in robotics*, Autonomous Robot Vehicles, vol. 8, pp. 167–193, 1990.

[34] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, MIT Press, 2006.

[35] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, H. Durrant-Whyte, *Simultaneous localization and mapping with sparse extended information filters*, International Journal of Robotics Research, vol. 23, no. 7-8, pp. 693–716, 2004.

[36] T. Tuytelaars, K. Mikolajczyk, *Local invariant feature detectors: a survey*, Foundations and Trends in Computer Graphics and Vision, vol. 3, no. 3, pp. 177-280, 2008.

[37] S. Umeyama, *Least-squares estimation of transformation parameters between two point patterns*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 4, pp. 376–380, 1991.

[38] G. van Meerbergen, M. Vergauwen, M. Pollefeys, L. van Gool, *A hierarchical symmetric stereo algorithm using dynamic programming*, International Journal of Computer Vision, vol. 47, no. 1, pp. 275–285, 2002.

[39] A. Vedaldi, H. Jin, P. Favaro, S. Soatto, *KALMANSAC: robust filtering by consensus*, IEEE International Conference of Computer Vision, pp. 633–640, 2005.

[40] O. Woodford, P. Torr, I. Reid, A. Fitzgibbon, *Global stereo reconstruction under second-order smoothness priors*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 12, pp. 2115–2128, 2009.

[41] W. Zhang, J. Kosecka, *Image based localization in urban environments*, 3D Data Processing, Visualization and Transmission, pp. 33–40, 2006.

[42] Z. Zheng, H. Wang, E. Teoh, *Analysis of gray level corner detection*, Pattern Recognition Letters, vol. 20, no. 2, pp. 149–162, 1999.

# Appendix A

# Recursive Bayesian estimation

The Bayes filter is used to recursively estimate a probability density function over time by using a state transition model and measurements. The filter is divided in two parts: the control update (prediction) and the measurement update (innovation). A prerequisite for using the Bayes filter is that the system is assumed to be a Markov process where all the information needed to predict the current state is contained within the previous state. Figure A.1 illustrates this Markov process as a graphical model with state vector $\mathbf{x}_t$, control input $\mathbf{u}_t$ and measurement $\mathbf{z}_t$.

Because of the generality of the Bayes filter there is no closed form solution with which it can be used. In this appendix we consider two approximations: the extended Kalman filter (EKF) [34, p. 54] and the particle filter [34, p. 96]. Both of these filters make a few assumptions about the system in order to simplify the Bayes filter and enable a closed form solution.

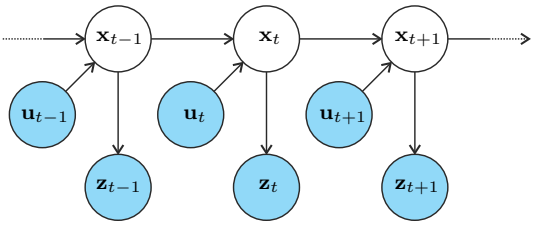



**Figure A.1** – Graphical model of a Markov process. The state vector $\mathbf{x}_t$ can be estimated using only the previous state vector $\mathbf{x}_{t-1}$, the new control $\mathbf{u}_t$, and the measurement $\mathbf{z}_t$.

## A.1 Extended Kalman filter

When a system and measurements made by the system's sensors can be described with linear equations, and the uncertainty in both the state vector and the measurement is Gaussian, the Kalman filter can be used. These limitations simplify the Bayes filter to a set of closed form expressions. The EKF uses first order Taylor approximations of a nonlinear system and measurement model to enable the use of the Kalman filter equations. The EKF is given in Algorithm A.1.

The first two lines of the algorithm form the control update. In Line 1 the state transition model $\mathbf{g}$ is used to update the state or mean vector $\boldsymbol{\mu}$ with the control input $\mathbf{u}$. Note that $\mathbf{g}$ can be a set of nonlinear equations. The next step, in Line 2, is to update the uncertainty of the state vector $\boldsymbol{\Sigma}$ with a linearized version of the state transition model. The linearization is done with the Jacobian of $\mathbf{g}$

**Algorithm A.1** EKF($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}, \mathbf{z}_t$)

1: $\boldsymbol{\mu}_t := \mathbf{g}(\boldsymbol{\mu}_{t-1}, \mathbf{u})$
2: $\boldsymbol{\Sigma}_t := \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^T + \mathbf{S}_t$
3: $\mathbf{K} := \boldsymbol{\Sigma}_t \mathbf{H}^T (\mathbf{H} \boldsymbol{\Sigma}_t \mathbf{H}^T + \mathbf{Q})^{-1}$
4: $\boldsymbol{\mu}_t := \boldsymbol{\mu}_t + \mathbf{K}(\mathbf{z}_t - \mathbf{h}(\boldsymbol{\mu}_t))$
5: $\boldsymbol{\Sigma}_t := (\mathbf{I} - \mathbf{K}\mathbf{H}) \boldsymbol{\Sigma}_t$
6: **return** $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

evaluated at the state estimate. The Jacobian is given by $\mathbf{G}$ and the covariance matrix of the process noise is given by $\mathbf{S}$. The process noise encapsulates the uncertainty in the state transition model.

The second part of the algorithm is the measurement update. We linearize the measurement function $\mathbf{h}$ using a Jacobian, and use it with the measurement noise covariance matrix $\mathbf{Q}$ to calculate the Kalman gain $\mathbf{K}$. The Jacobian $\mathbf{H}$ is evaluated at the state estimate. The Kalman gain is used with the error between the measurement and the measurement estimate, calculated with the measurement function, to refine the state vector estimate in Line 4. Finally we update the covariance matrix of the state estimate.

Ease of use and good performance make the EKF the first algorithm to consider for most estimation problems.

## A.2 Particle filter

When a system is highly nonlinear or the noise cannot be described adequately by a Gaussian distribution, the EKF can make large errors in estimation. To overcome such issues the particle filter is often utilized as it makes no assumptions about the probability density function to be estimated. Instead it describes the distribution with a Monte Carlo approximation, i.e. with particles (samples from the distribution). As these particles can be arranged in any fashion they can describe any probability function up to a certain resolution depending on the number of particles used.

The particle filter, given in Algorithm A.2, is also a simplified version of the Bayes filter and therefore follows the same structure as the EKF.

**Algorithm A.2** Particle_Filter($Y_{t-1}, \mathbf{u}, \mathbf{z}_t$)

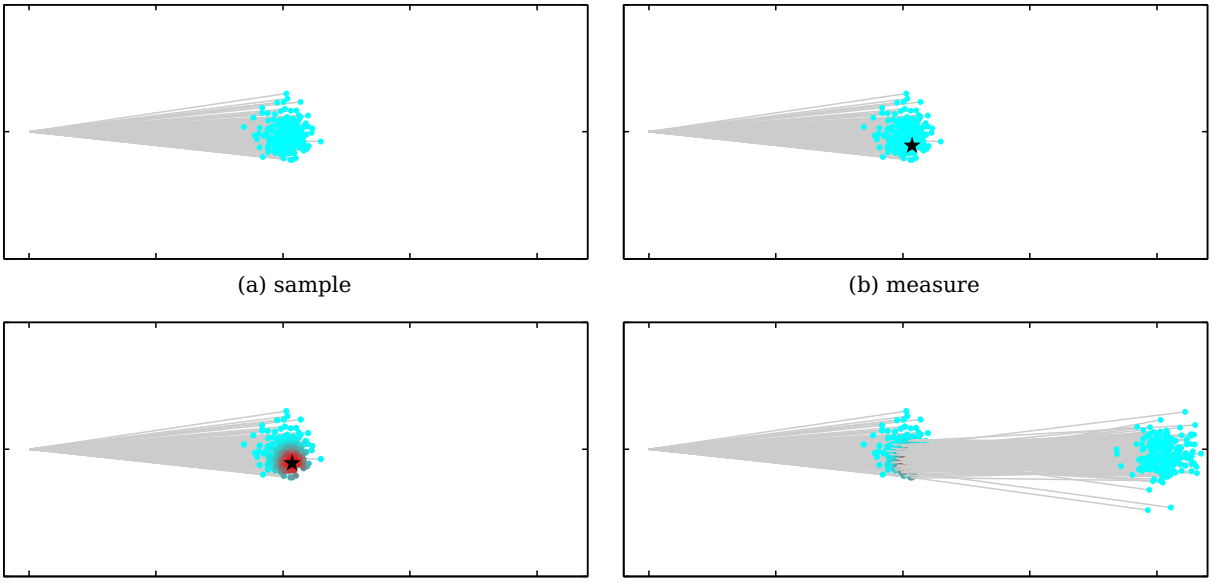
**for** all particles $k \in \{1, 2, \ldots, m\}$ **do**
  sample $\mathbf{x}_t^{[k]} \sim p(\mathbf{x}_t | \mathbf{u}_t, \mathbf{x}_{t-1}^{[k]})$
  $w_t^{[k]} := p(\mathbf{z}_t | \mathbf{x}_t^{[k]})$
**end for**
**for** all $i \in \{1, 2, \ldots, m\}$ **do**
  draw $k$ with probability $\propto w_t$
  add $\mathbf{x}_t^{[k]}$ to $Y_t$
**end for**
**return** $Y_t$

The first step of the particle filter is to enter a loop over all the particles. Within this loop we draw new locations for particles from the expected distribution, using the control command and the state

transition model. This is similar to the EKF control update. Using the measurement we calculate a weight (or importance factor) for each particle. The weight is calculated as the probability of the measurement given the particle in question.

Once the loop over all particles is done, we check whether it is necessary to resample the particles. This is done by normalizing the weights and by calculating the number of effective particles. If the number of effective particles is below a threshold, we resample the particles by letting some die and cloning others. New particles are drawn from the old set with a probability proportional to the weights of the old particles. A technique called stratified sampling is usually employed to do the resampling [34, p. 111]. If a particle has a high weight it is likely that it will survive in several new particles while particles with low weight are likely to die. For this reason the particle filter is often described as a Darwinian process, as it is a case of survival of the fittest.

In order to clarify the use of the particle filter we provide an example. Suppose we have a vehicle moving on a 2D plane. At every time step it receives a command and a noisy GPS measurement of its location. Figure A.2 (a) depicts the first step of the particle filter. The robot starts on the left side of the figure and moves to the right. The samples are drawn from the uncertainty attached to the estimation of this motion. The distribution of the samples represents the uncertainty in vehicle location after the control update. In (b) the vehicle's location is measured with the GPS. The particles are weighed by using the uncertainty in the measurement, depicted in (c). The final step is the resampling. Particles with high weights are resampled to be used at the next time step, while low weight particles are not used again. In (d) it is clear that particles close to the measurement survive for use at the next time step. We show the particle locations for the next time step, obtained by using the new control input and the resampled particle locations. The process can now be repeated at subsequent time steps.



(a) sample (b) measure

(c) weigh (red for high and cyan for low) (d) resample and sample new pose for the next time step

**Figure A.2** – Particle filter step by step.